HUMAN IN-PLACE ACTION RECOGNITION USING COMBINATION OF KINECT DATA STREAMS

BY

Mashaan Awad Alshammari

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

November 2015

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **MASHAAN AWAD ALSHAMMARI** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE.

Dr. Adel Fadhl Ahmed
(Advisor)

Dr. Lahouari Ghouti
(Member)

Dr. Abdulaziz Alkhoraidly
Department Chairman

Dr. Md. Rafiul Hassan
(Member)

Dr. Salam A. Zummo
Dean of Graduate Studies

DEANSHIP OF GRADUATE STUDIES

Date

To

my father, Awad, from whom I learned commitment
my mother, Nhaiah, from whom I learned persistence
my siblings, from whom I learned togetherness
my wife, Tahani, from whom I learned love
my son, Bakr, from whom I learned responsibility

ACKNOWLEDGMENTS

First and foremost all praise and glory to Almighty Allah

I wish to express my sincere gratitude to my advisor, Dr. Adel Fadhl Ahmed, Dean of the College of Computer Sciences and Engineering at King Fahd University of Petroleum and Minerals (KFUPM), for his endless support and guidance throughout my thesis research work. His kindness, insights, and enthusiasm motivated me during my thesis work.

I would also like to thank my thesis committee members: Drs. Lahouari Ghouti and Md. Rafiul Hassan for their priceless research advices and deep reviews that helped me reshape this work. Also, special thanks to my instructors during my course work at KFUPM: Drs. Sabri Mahmoud, Mohammed Alshayeb, El-Sayed El-Alfy, Faraj Azzedin, Uthman Baroudi, Sami Zhioua, and Musab Al-Turki; for their advices and thoughts that advanced my knowledge and research skills.

I would like to extend my thanks to my course projects partners: Loay Shabaneh, Salih Alyahya, Umair Yaqoub, and Salih Taqqiuddin, for their efforts and suggestions. Thanks to Ridwan Aljalali for his support and help in learning AutoDesk Maya.

Deep gratitude is expressed to King Fahd University of Petroleum and Minerals

(KFUPM) for providing the research facilities and tools to complete my research and course work. Thanks to my employer, University of Hail (UOH), for the financial support throughout my study at King Fahd University of Petroleum and Minerals (KFUPM).

Finally, I could not achieve this without the assistance of my family. I thank my parents, Awad Salem and Nhaiah Bsheet, for their endless sacrifice, sincere prayers, and honest advices throughout my life; my wife Tahani Fahd for her love, patience, and encouragement; my siblings for their help and being always there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME: Mashaan Awad Alshammari

TITLE OF STUDY: Human In-Place Action Recognition using Combination of Kinect Data Streams

MAJOR FIELD: Information and Computer Science

DATE OF DEGREE: November, 2015

Human action recognition is an important aspect of multiple real life applications (e.g., video search, health care, and human-computer interaction). The introduction of Microsoft Kinect© boosted massively this research field. Kinect defines and monitors a very restricted area for human movement detection. Human actions must take place in this restricted are in order to be captured and recognized accurately. This environment is only suitable for In-Place actions that occur in a single place without requiring the subject to move long distances. In this research, we aim to recognize In-Place actions such as punch, jump, or pick up. Traditionally, 2D video representations (i.e., Red- Green-Blue (RGB) and depth videos) were the main sources for human action recognition systems. However, the human

action features extracted from RGB and depth videos have major limitations such as background noise in RGB videos and self-occlusion in depth videos. The second release of Microsoft Kinect© SDK provides rich data streams, including: RGB frames stream, depth frames stream, body index stream, and body joint-tracking stream. In this thesis, we utilized depth frames stream, body index stream, and body joint-tracking stream to perform action segmentation, background subtraction and action recognition. We proposed a new feature set combining 15 features vectors, to recognize human actions recorded by Kinect system. The proposed features, extracted from joint positions, consist of: standard deviation, Euclidean distance, and discrete wavelet transform (DWT) metrics. Experiments were conducted using the proposed features and benchmarked against those based on raw joints positions. In experiments involving unseen subjects, the proposed features performed better than those based on raw joint positions. Moreover, generating a comprehensive training dataset of human actions is a challenging task. As a secondary objective for our research study, we investigated the feasibility of training a classifier on synthetic human action data and testing it using real human actions. Simulation results indicate that the classifiers trained entirely on synthetic data, attained 90% recognition rates on real test data, which supports our second objective.

ملخص الرسالة

الاسم الكامل: مشعان عواد سالم الشمري

عنوان الرسالة: التعرف على حركة الإنسان المكانية من خلال دمج البيانات المُلتقطة عن طريق جهاز مايكروسوفت كينيكت

التخصص: علوم الحاسب الآلي والمعلومات

تاريخ الدرجة العلمية: نوفمبر ٢٠١٥

تُستخدم تقنية التعرف على حركة الإنسان في تطبيقات عدة، من أهمها: التطبيقات الطبية وتطبيقات الحاسوب التفاعلي. تقديم مايكروسوفت لجهاز كينيكت© ساهم بشكل كبير في تطور هذا المجال البحثي. حركة الإنسان يجب أن تظهر في مساحة محدودة لكي يتم التعرف عليها عن طريق كينيكت. طريقة عمل كينيكت مناسبة للتعرف على حركة الإنسان المكانية التي لا تتطلب التحرك لمسافات طويلة. في هذا البحث، عملنا على التعرف على بعض الحركات المكانية مثل: القفز والركل. في السابق كان التعرف على حركة الإنسان يتم عن طريق تحليل لقطات الفيديو الملونة أو البُعدية. لكن هذه اللقطات تعاني من بعض الصعوبات عند استخدامها لوصف حركة الإنسان، مثل: لون الملابس في اللقطات الملونة أو تداخل أعضاء الجسم في اللقطات البُعدية. البرمجيات المُرفقة مع الإصدار الثاني من جهاز مايكروسوفت كينيكت تقدّم معلومات قيّمة، مثل: اللقطات الملونة واللقطات البُعدية ومؤشر الجسم والإحداثيات الثلاثية لبعض مفاصل الجسم. في هذه الأطروحة استخدمنا اللقطات البُعدية لتقسيم الحركة المكانية واستخدمنا مؤشر الجسم لإزالة الخلفية عن الجسم. وللتعرف على الحركة استخرجنا من الإحداثيات الثلاثية لبعض مفاصل الجسم ثلاثة أنواع من السِمات هي: الانحراف المعياري والمسافة الإقليدية بالإضافة الى بعض المقاييس المبنيّة على تحويل الموجات المنفصلة. النتائج المخبرية أظهرت تحسناً في أداء خوارزمية التعرف بعد تطبيق سِمات مُستخرجة من الإحداثيات الثلاثية لمفاصل الجسم مقارنةً باستخدام الإحداثيات الثلاثية لمفاصل الجسم كسِمات مباشرة. أيضاً من المشاكل التي تواجه تقنية التعرف على حركة الإنسان هي الحاجة الى كمية كبيرة من بيانات الحركة. تسجيل هذه البيانات الحركية باستخدام أشخاص حقيقيين يزيد من صعوبة العملية بسبب الحاجة إلى التواصل مع الشخص بالإضافة إلى شرح تفاصيل الحركة له. كهدف إضافي لهذه الأطروحة، درسنا إمكانية تدريب خوارزمية التعرف على بيانات حركية اصطناعية ومن ثم اختبارها على بيانات حركية حقيقية لتوفير الوقت والجهد. التجارب التي قمنا بها أظهرت أن خوارزمية التعرف التي تم تدريبها على بيانات حركية اصطناعية تمكنت من التعرف على بيانات حركية حقيقية بنسبة تعرف تجاوزت ٩٠% وهو ما يدعم هدفنا الثاني.

CHAPTER 1

INTRODUCTION

1.1 Overview

Human action recognition is an important problem in computer vision. Search results using the query "human action recognition" on a famous bibliographic database (Scopus), shown on Figure 1.1, demonstrate the growing interest in this topic. Despite its importance to the field of human computer interaction (HCI) [3], human action recognition has applications in other fields such as sign language [4], health care [5], and surgical training tasks [6].

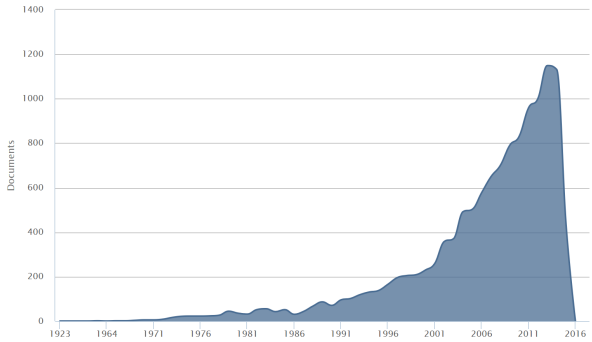


Figure 1.1: Number of human action recognition studies indexed by Scopus

To build a human action recognition system, a set of features characterizing the human action is required. Traditionally, the recorded Red- Green-Blue (RGB) videos are the only source for recognizing the human action. Existing techniques based on RGB videos for human action recognition, commonly use spatio-temporal interest points [7, 8], to capture the motion patterns. These interest points are accumulated into local distributions such as: histograms of oriented gradients (HOG) [9] or histograms of optical flow (HOF) [10], to represent spatio-temporal patterns.

It is difficult to extract accurate human action information from RGB videos [11]. This difficulty is attributed to the noise from the subject clothing and background objects. Moreover, tasks such as segmentation and silhouette extraction are complicated in RGB videos.

Depth images provided a breakthrough in the computer vision field [12]. Unlike RGB images, the pixel intensity in the depth image represents the distance from the camera to the first surface at the pixel location. Depth images are invariant to the scene color and lightening conditions. Subsequently, segmentation and silhouette extraction tasks became easier with depth images, which introduces new features to characterize the human action.

Existing depth image-based human action recognition algorithms, use a common preprocessing step, which is background subtraction as illustrated in Figure 1.2. As for the features, these studies followed two approaches. One approach is motion images and the other is spatio-temporal features. The motion image is

Figure 1.2: Depth Image with Subtracted Background

an accumulation of set of depth images, where the pixel intensity represents the motion history [13]. The human action recognition using motion images achieved by comparing the captured motion volume with already labelled data [14] or by pooling the spatio features [15]. The second approach of recognizing the human action using depth imagery is spatio-temporal features, where the features are quantized into a codebook [16, 17]. In the training codebook, the occurrence of the features is important whereas the features' order is disregarded. Although the depth images are invariant to color and lightening, they suffer from major limitations such as the need of large motions and the challenge of subjects self-occlusion [12].

Microsoft Kinect provides the body joints tracking system, which opens up another dimension to model and recognize complex human actions. Kinect uses pose estimation to extract human body joints from a single depth image. This method was proposed by Shotton et al. in 2013 [18]. The Kinect body joints tracking system estimates the approximate 3D coordinates (x, y, z) of selected human body joints considering the Kinect position as the origin of this coordinates

system.

Many studies utilized the human body joints tracking system provided by Kinect, to perform action recognition. Researchers used two methods to recognize the human actions using Kinect body joints tracking system:

- Some studies used raw joints positions to train the learning algorithms. The recognition stage, done by measuring the similarities between the test trajectory and the reference trajectory, then using thresholds to classify the test sample. The thresholds extractor can be: Dynamic Time Warping (DTW) [19], Hidden Markov Model (HMM) [20], or both [2].

- Other studies choose to extract features from the raw joints positions, then train the learning algorithm using those extracted features. These features can be histogram of oriented displacements (HOD) [21], joint angles and angular velocities [22], or directional velocity [23].

The movement of a human body joint in the Kinect coordinates system is a function that changes over time, also known as trajectory. Multiple features can describe body joints trajectories; this includes geometric and statistical features. In this work, we introduce a feature extraction techniques based on:

- Standard deviation,

- Euclidean distance, and

- Statistical analysis of Discrete Wavelet Transform (DWT).

Discrete wavelet transform (DWT) is a multiresolution analysis tool of one and two dimensional signals and functions. DWT is used to perform deep analysis for numerical values because they inherit the same properties of the original signal (i.e., the joint trajectory) over a finite interval. Moreover, Wavelet transformation maintains the temporal relations between numerical values.

Another area we attempt to improve in this work is the generation of human action data. Remarkably, a sophisticated action recognition system requires a comprehensive training dataset (e.g. different body builds and different Kinect angles). Nevertheless, generating the training dataset using real human subjects is an intensive task due to the communication time with the subject and the time consumed to explain the action for the subject. Additionally, noisy samples can occur during the recording sessions, which adds overhead to the recording process.

Recent computer graphics techniques are able to produce the inputs (e.g. RGB images, depth images, and human skeletons) for the computer vision algorithm. 3D computer graphics have been used to provide comprehensive training datasets for the learning algorithms, in many applications such as hand gesture recognition [24] and human pose estimation [18].

1.2 Motivation

Many studies in the field of computer vision tackled the problem of human action recognition. In spite of the intensive research, human action recognition is still a challenging problem. This challenge is due to the complexity of describing the

human action. There has been a lot of research to describe the human action using RGB and depth videos. Nevertheless, features extracted from RGB and depth videos have major limitations such as background noise in RGB videos and self-occlusion in depth videos.

The body joints tracking system provided by Microsoft Kinect is an invariant representation of the human body, which significantly eases the characterization of the human action. Nonetheless, using the joints positions (obtained by Kinect tracking system) as features of the human action might be vulnerable to the variations in the subject's behavior. Therefore, it is essential to investigate and explore new joints positions features to describe the human action.

Another issue related to the human action recognition is the need of adequate, comprehensive, and diverse training dataset. Many studies used real human subjects to record actions using Kinect, and then use them for training. However, this process is time and effort consuming, because the human subject needs a full understanding about the actions, which involves communication and explanation sessions.

On the other hand, computer 3D graphics provides an excellent simulation of the human subjects. Therefore, we investigate the feasibility of training the human action recognition classifier on synthetic human action data then used real human action data for testing.

1.3 Objectives and goals

In this work, we are aiming to explore new features out of the joints positions (obtained by Kinect tracking system) to describe the human action. Furthermore, we want to simplify the classification model by proposing new preprocessing techniques for the joints positions. Finally, we investigate the feasibility of using synthetic human action data to train the human action recognition classifier. To accomplish aforementioned goals, we act according to the following time line:

1. Obtain a standard human action data captured by Kinect from a recent study in the literature [2].

2. Compare the newly proposed features with the approach followed by baseline study [2], to verify the usefulness of the new features.

3. Use 3D graphics software to generate the training human action data, in addition to recording Kinect human action data to be the test data.

4. Apply the new features to synthetic data as well as Kinect data, and investigate the feasibility of training with synthetic data.

5. Repeat the classification process with different schemes of the new preprocessing techniques, to determine the efficiency of these techniques.

1.4 Contributions

This thesis introduces new features extracted from the joints positions. In addition, it presents a new preprocessing step for the joints positions, in order to simplify the classification model. Lastly, it presents an efficient human action recognition system that entirely trained on synthetic human action data. Our contributions in this thesis listed in the following bullets:

- Describing the amount of change in the body joints position by extracting the standard deviation of its positions over time. The experiments demonstrated that this was a good descriptor for human action, especially when it combined with the statistical properties of discrete wavelet transform DWT.

- Measuring the distance traveled by the body joints using 3D Euclidean distance as well as the maximum 1D Euclidean distance.

- Supported by our experiments, applying DWT to the original trajectories of the body extremities joints proved to be a good descriptor for the human action. We selected 8 statistical properties to capture the linear behavior as well as nonlinear behavior of the signals obtained by DWT.

- We introduced a new preprocessing step, called quantization. In quantization, the positions of the body joints assigned into predefined groups represented by their centers, this step simplified the classification model.

- We achieved recognition rates above 90% on real human action data using classifiers that were entirely trained on synthetic human action data. This

demonstrated the feasibility of using synthetic human action data to train the human action recognition classifier. Using synthetic data, we can simulate many scenarios that are difficult to implement in reality such as: Kinect installation locations and wide range of human subjects.

1.5 Outline

For the rest of this thesis, background about related topics is given in Chapter 2. Overview of related work is listed in Chapter 3. The proposed approach (including datasets generation, preprocessing, and features extraction) discussed in Chapter 4. Chapter 5 introduces experiments, results, and analysis. Finally, conclusions and findings presented in Chapter 6.

CHAPTER 2

BACKGROUND

2.1 Microsoft Kinect

The Kinect [25] was released by Microsoft on November 4, 2010 [26]. It breaks the previous record of *"Fastest-selling gaming peripheral"* by selling 8 million units in the first 60 days [27]. Previously, Playstation EyeToy© was a breakthrough in the gaming market. It provides an action-based game control instead of traditional controllers; this innovation earned it market domination for a few years. However, the EyeToy needs constant light for proper operation, which was a major shortcoming. The Kinect overcomes this deficiency by providing high quality depth images that makes it operate efficiently under different positioning and lightning situations. Moreover, the Kinect shipped with sophisticated machine learning techniques that provide powerful pose estimation that leads to a smooth body joints tracking.

Primarily, Microsoft started the development of Kinect in June 1, 2009 under

the name of "Project Natal" [28]. Thereafter, the Kinect name was derived from 'kinetic' and 'connect'. Kinect is available for the end users in three ways:

- Kinect for Windows,
- Kinect for Xbox as an external supplement, and
- Bundled with the Xbox.

Kinect is not only a gaming device as it provides high quality color and depth images for research community with an affordable price. In 2011, Microsoft released the Kinect SDK bundled with tools and APIs (Application Programming Interfaces) to develop Kinect applications [26]. The release of Kinect SDK in addition to already existing libraries (e.g. OpenNI and OpenKinect) made the Kinect more popular for computer vision research.

The Kinect belongs to the category of RGB-D sensors. These types of sensors provide a color image as well as estimated depth of every pixel in the scene. Kinect is able to capture 30 frames per second (FPS) for both depth and color images. Kinect includes advanced sensing hardware which are [29]:

- **RGB camera**: captures color images in 1080p resolution.
- **Infrared (IR) emitter**: emits infrared light beams.
- **Infrared (IR) depth sensor**: receives infrared beams reflected back to the sensor to generate 512×424 depth images.
- **A multi-array microphone**: contains four microphones to capture sound. Since there are four microphones, it is possible to locate the sound source.

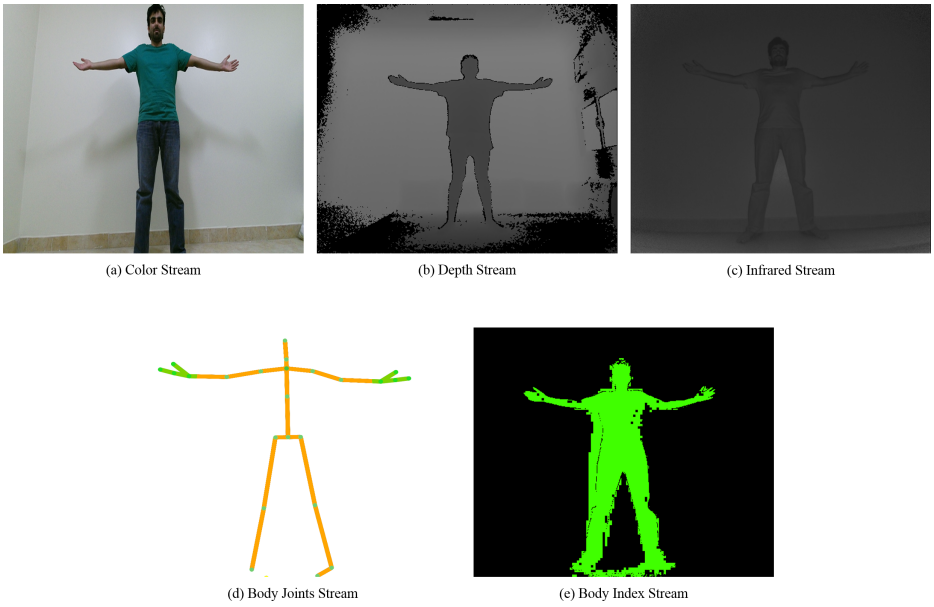


Figure 2.1: Kinect v2 Different Data Streams

These internal components of the Kinect are combined to produce data streams shown in Figure 2.1:

- **RGB stream,**
- **Depth stream,**
- **Infrared stream,**
- **Body joints positions stream, and**
- **Body index stream.**

Kinect uses pose estimation to extract body joints. The second release of Kinect supports the track of 25 joints. The joints are: (Head, Neck, SpineMid, ShoulderLeft, ElbowLeft WristLeft, HandLeft, ShoulderRight, ElbowRight, WristRight, HandRight, SpineBase, HipLeft, KneeLeft, AnkleLeft, FootLeft, HipRight, KneeRight, AnkleRight, FootRight, SpineShoulder, HandTipLeft, ThumbLeft, HandTipRight, ThumbRight) [29].

2.2 Pattern Recognition in Brief

In pattern recognition, observations (patterns and regularities) are classified into predefined categories based on their properties [30]. Pattern recognition has many applications such as speech and text recognition. Figure 2.2 shows the structure of a typical pattern recognition system [31].

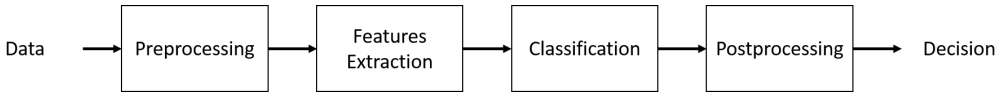


Figure 2.2: Structure of typical pattern recognition process

Features extraction and classification methods contribute significantly to the overall performance of any pattern recognition system, therefore, we discuss them in the next two subsections. In the final subsection, we introduce the performance metrics used to evaluate pattern recognition systems.

2.2.1 Features Extraction

Features extraction is a crucial part of any recognition system, since it directly influence the performance of the classifier. The goal of features extraction process is to characterize the observations using measurements. Ideally, the difference between measurements values are small for observations within the same class (intra-class) and large for observations from different classes (inter-class) [31]. A feature can be numerical value (e.g., height of a person) or categorical value (e.g., gender).

2.2.2 Classification Methods

In machine learning, a classifier is the learning algorithm that used to classify new instances into predefined categories. Initially, labelled data is fed to the classifier to earn the ability of prediction, usually this is called the training process which is illustrated in Figure 2.3. In this section, we introduce the principles as well as historical background of the classifiers we used in this thesis.

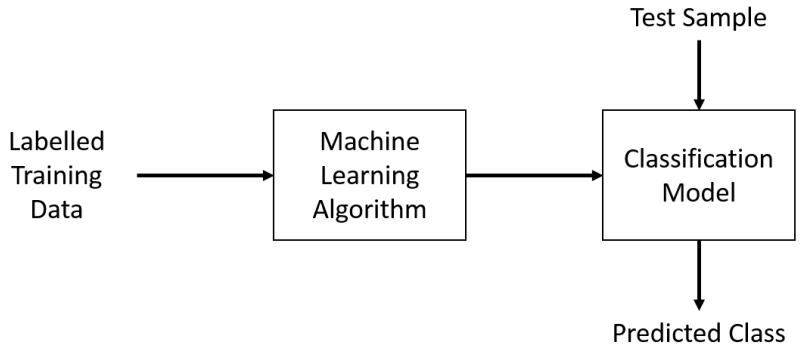
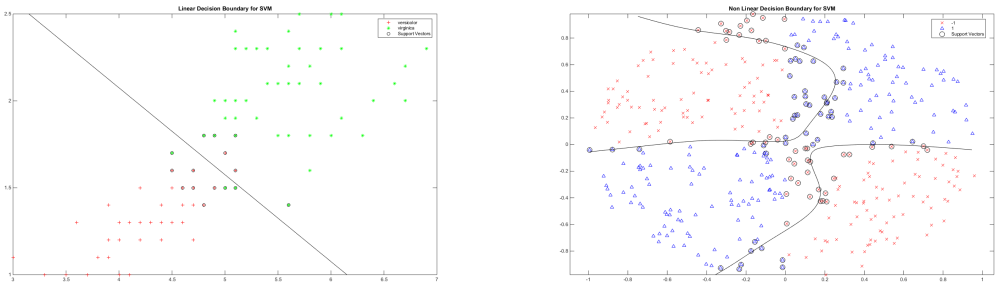


Figure 2.3: Classification Process

Support Vector Machine

Support Vector Machine (SVM) depends on representing data on higher dimensions that is usually higher than original feature space. The mapping is done by linear or non-linear function (illustrated in Figure 2.4) and the support vectors determine the margins. A major advantage of SVM approach is that the complexity of resulting classifier totally depends on the support vectors; not on the transformed space. This makes SVMs less prone to over-fitting problems compared to other methods [31]. The principle of SVM was developed by Cortes and Vapnik in 1995 [32]. Furthermore, SVM is a powerful classification method it proved



(a) Linear SVM (b) Non-linear SVM

Figure 2.4: Linear and non-linear SVMs [1]

to be excellent performer in a wide range of applications, such as handwritten recognition and categorization of Web pages [33].

Random Forest

Random Forest (RF) is an ensemble learning technique. Usually ensemble learning algorithms provide more accurate results than single classifiers since they are robust to some confusing problems such as noise and samples correlation [34]. RF is constructed by multiple randomized decision trees. Figure 2.5 demonstrates the basic structure of decision tree classifier. The randomized trees in the forest receive the sample as an input vector then predict its class. The predicted classes are collected and the final decision is made based on majority voting. Some advantages of RF classifier include tackling large datasets, estimating the features that influence the decision and its strength in estimating missing data [34]. The fundamental basis of the RF was first introduced in 1997 by Amit and Geman [35]. Their methodology was based on searching over a random subset of decision trees after splitting the node. However, Random Forest was properly introduced in

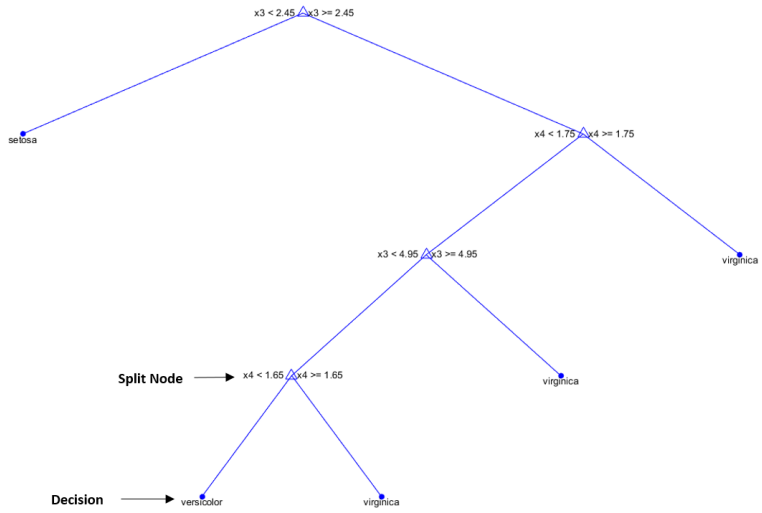


Figure 2.5: Typical decision tree classifier

2001 by Breiman [36] where he explained well how to build a forest of uncorrelated randomized decision trees.

k-Nearest Neighbor

k-Nearest Neighbor (k-NN) is a non-parametric classifier which operates based on the k-NN classification rule. k-NN classification rule recognizes an unlabeled sample x by assigning it the label of the majority samples among k samples [31]. k-NN classifier known as one of the simplest and fundamental algorithms in machine learning. Before the classification phase the user has to define the value of k. Usually, k-NN classifier used two distance functions: Euclidean distance and Manhattan distance. The principle of k-NN classifier firstly introduced by Fix and Hodges Jr in 1951 when they proposed a non-parametric discernment analysis [37]. It took almost 20 years for computer technology to advance in order for researchers to apply the theoretical foundations.. k-NN introduced in its

computational version by Cover and Hart in 1967 where they establish the k-NN classification rule [38]. Moreover, their job have been taken further by Devroye in 1981 where he analyzed the discrimination of k-NN [39].

2.2.3 Evaluation Measures for Classification

Confusion matrix shown in Table 2.1, is a standard output of any classification system. The values in the diameter of the confusion matrix are the correctly classified instances, while other values represent the misclassified instances. The classification measures are extracted from the confusion matrix. In this work, we used two measures; Accuracy and F-Measure.

Table 2.1: Confusion matrix for two-class classification

		Predicted	
		Positive Class	**Negative Class**
Actual	**Positive Class**	True Positive (TP)	False Negative (FN)
	Negative Class	False Positive (FP)	True Negative (TN)

Accuracy

Accuracy is the most intuitive measure for the confusion matrix. It calculates the percentage of the correctly classified instances as shown in Equation 2.1.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.1}$$

Nonetheless, accuracy is not appropriate metric to evaluate imbalance-learning systems [40]. For example, if we have two classes where 95 instances in the first

class and only five instances in the second class. Apparently, a trivial classifier can get 95 accuracy by simply assigning the first class to all 100 instances.

F-Measure

The deficiency of accuracy measure leads to proposing new metrics geared towards evaluating the classifier performance for each class independently. Precision (Equation 2.2) and recall (Equation 2.3) are famous per class evaluation metrics [41]. Giving a particular class, precision measures the percentage of correctly classified instances over all instances in actual class, whereas recall measures the percentage of correctly classified instances over all instances in predicted class.

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

F-Measure (Equation 2.4) is the harmonic mean of precision and recall [42]. F-Measure thoroughly examines the classifier performance at a particular class.

$$F\text{-}Measure = \frac{2 \times precision \times recall}{(precision + recall)} \tag{2.4}$$

2.3 Human Action Recognition

Human action recognition is a process performed by the computer to interpret any action made by the user [43]. Nevertheless, a mouse click or a head scratch can be considered as an action by the previous definition. Mainly, sensing devices designed to capture specific motions by the user in order to maintain convenient usage. For example, some devices try to capture facial expression (e.g. a smartphone that decrease the volume when the user looks away).

Automated human action recognition is not a new concept. In 1963, RAND Corporation released the RAND tablet, which was the first pen-input device The project was funded by Advanced Research Projects Agency (ARPA) [44]. RAND tablet allows users to make free hand drawing using the stylus. The stylus emits pulses to an electrical grid that tracks the stylus movement to draw on the screen.

Throughout the development of action recognition, many devices served as input devices for the recognition algorithms. Wired sensors used heavily to detect human actions and gestures. Regardless of their superb accuracy, the idea of wearing wired sensors (e.g. gloves or suit) is not convenient.

Moreover, interactive environments [45, 46] had been developed to mimic the subject actions [47]. Maes et al., developed in 1995 a live mirror-based system called Artificial Life Interactive Video Environment (ALIVE) that enables subjects to simulate a drawn models on the screen [45]. The authors designed (ALIVE) to outperform the virtual reality systems, which uses goggles and gloves to simulate the virtual environment. Regardless of the beauty provided by virtual environ-

ment, their detection ability is questionable; because they just reflect the subject status and the graphical models represented the artificial part.

Another technique for action recognition is controller-based recognition systems, where the user holds a controller that is tracked by a sensor to interpret the human action. A very famous controller is the PlayStation Move Controller by Sony Computer Entertainment [48]. The PlayStation camera tracks where the player holds the controller.

Nevertheless, wearable sensors or controllers might restrict the user natural movement, in addition of being invasive and expensive [49]. Consequently, several studies investigate the vision-based recognition systems. Sony Corp. made a huge success when they release their EeyToy [50] in 2003. The 2D EyeToy was a peripheral for Sony Playstation 2, it translated the body movements into game controls. Nonetheless, the EyeToy has a severe limitation since it requires a certain amount of light in order to see the player.

Depth images are special type of images, where the intensity of a pixel represents the distance from the camera to the first surface at the pixel location. Depth images can serve as an input for action recognition algorithms; they provide an approximate 3D representation of a user body that can be used to detect particular actions. Microsoft Kinect is a famous example of action recognition using depth images. Kinect provides a high quality color and depth images for research community with an affordable price. Traditionally, 2D videos (including RGB and depth) were the only source for recognizing the human action. Nevertheless,

the body joints tracking system provided by Kinect opened another dimension for computer vision researchers to model and recognize complex human actions.

Recent computer graphics techniques are able to produce inputs (e.g. RGB images, depth images, and human skeletons) for computer vision algorithms. Using computer graphics, designers can produce high quality images to mimic the real world (similar to what we see in video games and movies). In the next section, we introduce an overview about computer graphics and how human characters can be realistically modeled and animated in 3D graphics software in order to serve as a source for input data to computer vision algorithms.

2.4 Computer Graphics

Computer graphics is a sub-field of computer science discipline, which studies synthesizing objects in digital world using some mathematical representation. This broad definition contains three tracks: 3D computer graphics, 2D graphics and image processing. Nonetheless, the term Computer graphics usually refer to 3D computer graphics. 3D computer graphics has three stages:

- **3D modeling:** The developers design 3D models (human, car, table, etc...) which are a mathematical representations based on a collection of geometric primitives (e.g. cube, sphere, and pyramid). Also, the mathematical representation can be constructed from a collection (also known as mesh) of basic geometric shapes such as polygons, triangles, curves. The idea behind representing 3D models as geometric shapes is to make them suitable for

graphical computations [51].

- **Smoothing and Animation** Once the models are ready, the developer has to define the deformation of the 3D object over time. This can be achieved using well-known motion techniques such as: key frames, inverse kinematics (IK), and motion capture (MoCap).

- **3D rendering:** is the process of generating 2D images based on 3D models in the scene. 3D rendering encompasses projecting 3D models (perspective), visibility handling (which part of 3D model is viewable? which part is not?). 3D rendering can be an intensive computational task because complex lightning models and shading/coloring algorithms may be involved in the scene.

Designing human characters is one of the core applications of 3D graphics. Despite its essential use in gaming industry and the movie entertainment industry, human 3D modeling emerged to be a fundamental aspect in medical and artificial intelligence disciplines. There are numerous software tools for 3D computer graphics, the famous ones are: Autodesk 3ds Max, Blender, Autodesk Maya, and Cinema 4D. These software tools provide the needed tools to create 3D human models. The steps needed to create a human model are:

- **Create the body mesh:** drawing the human body using geometric primitives (e.g. cubes and spheres).

- **Rigging the body mesh:** generating a human skeleton by mimicking a set of joints.

- **Skinning the mesh:** binding the human rig on top of the human mesh.

- **Adjusting joints influence:** defining how each joint affects the deformation of the skinned mesh.

Nevertheless, drawing and rigging a human body from a scratch using 3D software is not a trivial task (usually done by 3D designers). Moreover, it can be more complicated if too much details of the human body are needed (such as muscles). Fortunately, there are software to automate some of the aforementioned steps of drawing a human body. To generate a human models that are rigged and skinned, a developer can use software packages such as DAZ 3D (`http://www.daz3d.com/home`) or AutoDesk character generator (`https://charactergenerator.autodesk.com/`) shown in Figure 2.6 which is an online tool to generate 3D human models. Furthermore, tools in AutoDesk Maya 2015 (`www.autodesk.com/products/maya`) can also be used to control the human models and to animate them. AutoDesk Maya provides an innovative tool for human animation (Autodesk HumanIK). Autodesk HumanIK (Figure 2.7) used to target and solve full-body inverse kinematics (IK). After drawing the human rig, Autodesk HumanIK imports the rig and retargets a human model in order to produce a natural human deformation.

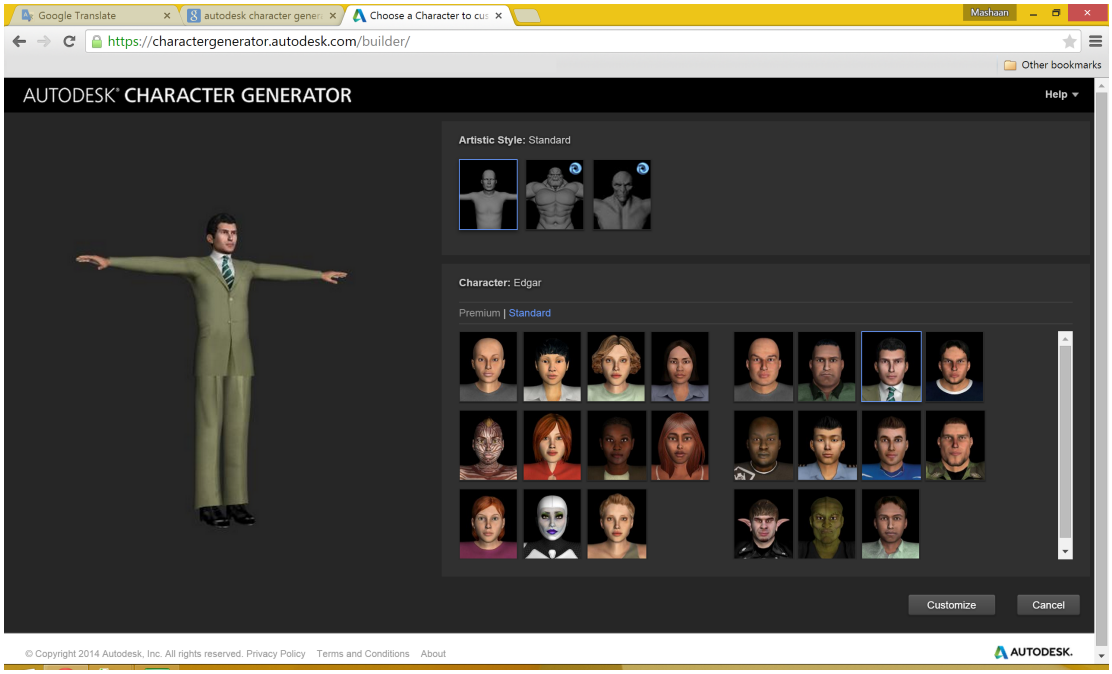


Figure 2.6: Character selection window in AutoDesk Character Generator

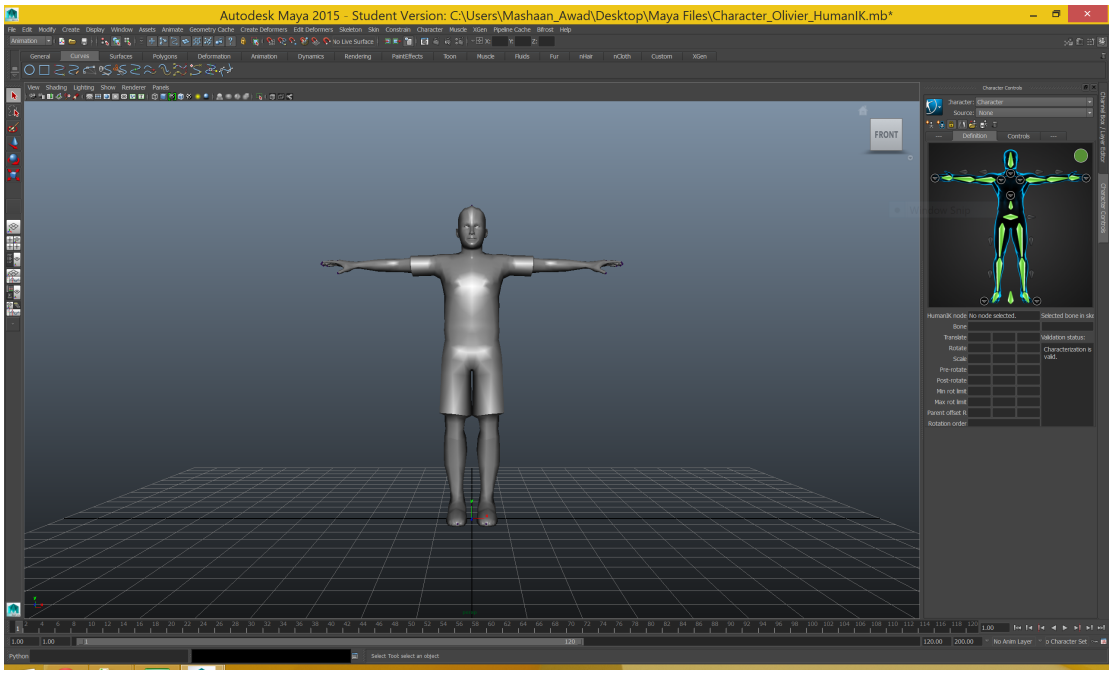


Figure 2.7: Human character and its HumanIK representation in AutoDesk Maya

CHAPTER 3

LITERATURE REVIEW

In 2013, Han et al. introduced a taxonomy of the computer vision topics that have been investigated by different researchers in computer vision using Kinect (as illustrated in Figure 3.1) [52]. The study identified four computer vision areas, that enhanced by the introduction of Microsoft Kinect. The first area is object tracking and recognition. Studies that addressed the problem of object tracking were interested in the exact position of the 3D object [53, 54], while the main objective of object recognition is to classify the object regardless of its position [55, 56, 57].

The second area of the taxonomy is human activity analysis, which encompasses pose estimation [18, 58, 59] and activity recognition [2, 19, 22]. The third area of the survey is hand gesture analysis, which includes hand detection [60], hand pose estimation [61], and hand gesture classification [24]. The last area covered by this survey is indoor 3D mapping, which aims to digitalize the indoor environment to ease tasks like localization and navigation [62, 63].

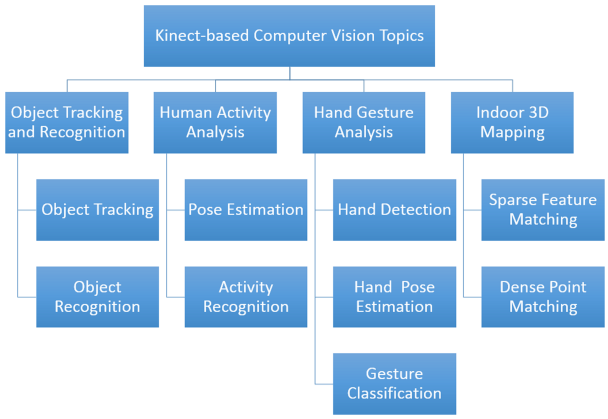


Figure 3.1: Computer Vision Topics Investigated using Microsoft Kinect

This thesis falls into the second area of the previous taxonomy (i.e., human activity analysis) [52]. Studies that address the action recognition problem have two types of contributions: contribution in the recognition method and/or contribution in the features extraction process. One of the core aspects in action recognition is the features extraction process. It is a decisive process on the results obtained by the recognition system. Different studies adopted various features extraction techniques. This diversity in features extraction techniques is due to the following factors:

- Nature of the desired gestures, and how to extract the most representing features [11].
- Computational cost of extracting the features, most of the studies tend to use cheaply extracted features in order to extend the system to automated recognition [64].

In the upcoming sections, we discuss how researchers used Kinect data streams to tackle the human action recognition problem.

3.1 Human Action Recognition using RGB-D Features

In this subsection, we explain how different studies employed RGB-D features to recognize hand gestures or body action.

In 2013, Zhang and Feng proposed a new approach of dynamic hand gestures recognition [65]. The proposed identification approach consists of two stages: frame fusion with density distribution features (rough recognition) and hausdorff distance or fingertip detection (accurate recognition). Initially, the framework mapped the dynamic hand gesture into a static fusing frame images, then the Density Distribution Features (DDF) was used to estimate the collection where the gesture belongs to. For the accurate recognition, hausdorff distance or fingertip detection performed on the last frame of the gesture to predict the precise gesture. The dataset was collected using ordinary camera and contain 10 dynamic grasp gestures. The experimental results showed that the proposed approach outperformed the single frame recognition and kept a consistent accuracy above 90% for all classes. Nevertheless, the study used samples collected from ordinary camera under the condition of constant light. Such condition and constraints make the approach not applicable for automated detection.

Dinh et al. introduced a hand gesture recognition system for smart home appliances in 2014 [24]. The proposed method relied on recognizing the hand parts then the state of each finger, because this will decide the gesture. The gesture recognition was performed by two stages. Firstly, the hand parts were recognized

using Random Forests (RFs) [36] classifier which was trained using thousands of synthetic depth maps. Secondly, the hand gesture was recognized based on the fingers' states detected in the first stage. The proposed method scored 98.50% as a mean of offline recognition rate for 400 samples representing four hand gestures from five subjects.

In a like manner, Dinh et al. introduced a hand number gesture recognition in 2014 [66]. The hand number recognition was achieved by recognizing hand parts in depth images. The number gesture recognition process executed in two parts: hand parts recognition by random forests (RFs) [36] and number gestures recognition by rule-based classification. The first part was trained using synthetic data to provide sophisticated hand parts recognition. Then the rule-based classification checked the state of the finger in order to detect the accurate number gesture. The proposed method showed a promising performance on a dataset of 500 samples representing 10 hand number gestures from five subjects (100 each). Both of the previous studies follow a similar approach which is training the RF classifier on a synthetic data then perform the recognition. However, these studies rely on the depth silhouette (i.e., hand depth image with subtracted background) of the hand which can be noisy representation of the hand.

Keskin et al. introduced a real time hand posture capture in 2013 [67]. The intended method performed object recognition by parts using depth images provided by Kinect. The human hand was represented using 21 different parts. The intermediate parts recognition was performed using multiple randomized decision

trees (RDT) [36] to identify the hand part for each pixel. Thereafter, a mean shift algorithm implemented to predict the joint centers. Finally, SVM classifier employed to detect the hand gesture. The authors tested their approach using the ten digits of American Sign Language (ASL). Initially, they trained the random forests on synthetic images of ASL digits. Then, the real depth images were fitted into trained hand skeletons. Finally, the resulted gestures were classified using support vector machine (SVM) [68]. The real data was collected using Kinect from 5 different persons where each person simulates the ten ASL digits. The dataset contains 300 frames for each digit per person resulting 15,000 images. Nevertheless, the proposed approach achieved 99.9% using SVM on real data collected by Kinect.

In 2013, Wang et al. introduced a real-time human action recognition framework to recognize speaker's gestures using Kinect depth images [15]. Usually, depth data are noisy due to limited measurement accuracy of depth sensor. This study proposed a robust background subtraction method where the pixels' depth values are represented as a Gaussian distribution. Once the human body appears in the frame the significant pixels have a small probability to belong to the background. Moreover, a Potential Active Region (PAR) was implemented in this study to minimize the computational complexity. The subject arm won't be detected if it was outside the PAR. Once the arm was observed in the PAR the hand movement got recorded and represented as Motion History Images (MHI). The classification process performed on two stages. An initial classification performed

by multi-class SVM model to classify MHI frames into movements. Then, the final classification performed by a set-based soft discriminative model to recognize the action (i.e., up, down, go, back and click) based on the movements. The authors used Kinect to generate a dataset that contains 150 gestures where 60 gestures were used for training and the remaining gestures were used for testing. The experimental work showed the efficiency of the proposed approach (90% accuracy) due to MHI features and soft discriminative model. This study contains some innovative techniques in order to recognize gestures from depth images only. However, restricting the speaker to a specific region to recognize the action might be considered as a constraint that make the method a bit controlled.

Anupam et al. developed a system to recognize the activity involving two persons using Kinect depth images [69]. The type of depth features were either shape or temporal features. For the shape features the RoI (Region of Interest) around the silhouette were divided into 8×5 and a histograms were obtained to represent the pixels' values in each box. As for temporal features, the difference frame between two consecutive frames was extracted then the fraction of white pixels was calculated. SVM classifier with polynomial kernel used to classify nine activities involving two persons. The classifier got high accuracy on actions: fighting, passing bottle, Japanese greeting, one drops the other picks, sit and talk, and handshake. However, the classifier showed a moderate performance on actions: bye waving, one reads the other stands, and one walks to the other.

Table 3.1: Summary of studies reviewed in Section 3.1

Author	Year	Application	Data	Features	Machine learning Technique
Zhang and Feng	*2013*	Hand gesture recognition	Using ordinary camera, 10 dynamic grasp gestures were collected.	Density distribution features (rough recognition) and hausdorff distance or fingertip detection (accurate recognition).	Fusing frame images recognition.
Dinh et al.	*2014*	Hand gesture recognition	400 samples representing four hand gestures from five subjects.	Hand depth silhouette to recognize hand parts. Recognized hand parts were used to identify the gesture.	Hand parts recognition using RFs. Recognition rules to determine the gesture.
Dinh et al.	*2014*	Hand number gesture recognition	500 samples representing 10 hand number gestures from five subjects (100 each).	Hand depth silhouette to recognize hand parts. Recognized hand parts were used to identify the gesture.	Hand parts recognition using RFs, recognition rules to determine the number gesture.
Keskin et al.	*2013*	Real time hand posture capture	5 different persons where each person simulates the ten ASL digits using Kinect.	Depth intensity differences of two pixels.	Randomized decision trees (RDT) to identify the hand part for each pixel. SVM to recognize number gesture.
Wang et al.	*2013*	Presenter action recognition	150 gestures where 60 gestures used for training and the remaining gestures used for testing.	Motion History Images (MHI).	SVM model to classify MHI frames into movements. Then, a set-based soft discriminative model to recognize the action.
Anupam et al.	*2012*	Two persons activity recognition	9 activities (5 videos each).	Histograms of RoI (Region of Interest) around the silhouette.	SVM with polynomial kernel.

3.2 Human Action Recognition using Body Joints Positions Features

Body joints tracking stream provided by Kinect started a new dimension of human action recognition. In this subsection, we inspect how researchers employed body joints tracking stream in order to recognize human actions.

Chang developed a Kinect human action recognition system to control PowerPoint slides in 2013 [23]. In this study SVM classifier was employed to perform online segmentation of the gestures then the frames passed to hidden Markov Model (HMM) classifier to recognise the speaker gestures (backward scroll, forward scroll, and pointing). The study pulled the upper body joints world positions

from Kinect then change their centroid to be the hip center, then the positions are normalized using minimum and maximum values in the training set. Moreover, the study used the directional velocity of the joints as an additional features to discriminate the actions that uses the same space but with different directions.

Another feature representation was developed by Saha et al. in 2014 where they inspected emotional action recognition. Mainly, in this study they used geometric features where they considered the hands distance with respect to spine. Moreover, the angles between joints was also involved in the features vector. Two angles were used: angle of (head, shoulder center, spine) and angle of (shoulder, elbow, and wrist). The study focused on recognizing five emotions from the body gestures, which are: Anger, Fear, Happiness, Sadness, and Relaxation.

Arici et al. developed a Kinect action recognition system that used a weighted Dynamic Time Warping (DTW) [71] with innovative preprocessing feature extraction in 2014 [19]. Basically, the method measures the similarity of joints world positions in the sample action and the reference action to make a decision. The preprocessing consists of three stages:

1. Translating the test sample into the middle of the scene by subtracting the hip center from all other positions.

2. Draw two orthogonal vectors from the shoulder center then uses three rotation matrices (one for each axis) to rotate the test sample to be orthogonal to Kinect view.

3. Normalizing the test sample by the distance between left and right shoulders.

Additionally, the authors addressed a deficiency of regular DTW, that it gives equal weights to all joints dimensions regardless of their contribution to the gesture. Therefore, the study introduced a weighted DTW where the weights determined by the joints contribution to the gesture. The joint contribution calculated based on its total displacement during the training stage. The authors collected the dataset from 38 participants that performed 12 gestures with 6 samples per action class. The study demonstrated how the preprocessing and weighted DTW enhance the recognition rate (96.64%) comparing to (62.41%) scored by regular DTW.

In a similar manner, Ibañez et al. In 2014 developed a action-recognition known as Easy Gesture Recognition (EasyGR). EasyGR is a tool to be used by developers with minimum knowledge of machine learning algorithms [2]. EasyGR allows the developer to create his own training dataset in order to recognize the desired gestures, this releases ordinary developers from machine learning details and optimal parameters. Mainly, EasyGR supports two learning algorithms: DTW [71] and Hidden Markov Models (HMM) [72]. The authors evaluated the proposed tool by investigating two metrics:

- **Accuracy:** recognition rate of specific actions.
- **Development effort:** effort needed to develop action-controlled applications with/out EasyGR.

DTW and HMM require training dataset to recognize testing gestures, the EasyGR user must manually segment the training frames and check their cor-

rectness. As a proof of concept, the developers constructed a training dataset of seven gestures (Circle, Elongation, Swim, Smash, Punch, Swipe Right, and Swipe Left). The training dataset contains 560 samples collected by 4 persons simulating aforementioned gestures 20 times. Using 10-folds cross validation on the collected dataset, DTW and HMM achieved offline recognition rates over 99%.

Furthermore, the study emphasizes on the simplicity of developing using EasyGR by measuring the design effort of developer with/out EasyGR. Ten developers were asked to develop an application controlled by the 7 gestures mentioned above using: rule-based approach and EasyGR approach. Two metrics were used to measure the effort: non-blank, non-commented lines of code (NLoC) and total time to implement a given action. The experimental results showed a significant reduction in average NLoC and average time using EasyGR, therefore it meets the simplicity aspect set at the beginning of the study.

Wang et al. developed a recognition system in 2013 to recognize two hand gestures [73]. In this system, the preprocessing step utilized the body index stream provided by Kinect SDK. Simply, this stream identifies the pixels that belongs to the human body. Mainly, the preprocessing aims to segment the hands from other parts in the image, by drawing a square based on the hand x position and monitor the movements in the coming frames. The features are a collection of six-vectors representing the world positions of the hands. The six-vector obtained from previous frame was subtracted from the six-vector of the current frame then it was multiplied by the frame rate to obtain the velocity. A Coupled HMM

model was used for recognition, where HMM chain represents a possibility of a hand trajectory. The model was tested on 8 two-hand gestures performed twice by 10 subjects. Using 60% of the data for training, the model scored an average accuracy rate of 92.87%.

Sharaf et al. introduced a real-time system for action detection in 2015 [22]. The system used joint angles and angular velocities derived from 3D joint world positions. The study used two types of features: probability distribution of skeleton features and temporal pyramid construction to describe the temporal information. MSRC-12 dataset and one action from G3D dataset were used to test the model that scored 91.1% and 93.7% respectively.

Gowayyed et al. in 2013, proposed a novel descriptor for 2D trajectories which is Histogram of Oriented Displacements (HOD) [21]. Basically, the trajectory of the joint was projected on 2D plane that represents a pair of dimensions (i.e., XY, XZ, and YZ projections). Afterwards, the trajectory was described using a histogram of directions between each two consecutive points. The method provided a discrimination power which was reflected of the experimental findings.

Acorn et al. developed a recognition system (in 2015) for sit-to-stand movement [74]. The study aims to identify the subjects who have a back pain that affects their normal sit-to-stand movement. The data was collected using Kinect sensor in a controlled environment (i.e., constant lighting and no movement in background). The action was performed three times facing the Kinect, in addition to three times with 45° off the Kinect. Thirty-one subjects (21 men and 10 women) volunteered

to record their actions which produced 93 samples. Moreover, a faulty sit-to-stand actions were performed by three subjects which produced 24 samples. Afterwards, the features were extracted from dynamic time warping of time series created by the joints movements over time. Two classification models were built based on: Random Forests and k-Nearest Neighbors. Experimental outcomes indicate a good results of both classification models.

Advances in other pattern recognition fields was reflected on computer vision studies. Le et al. adopted recent advances in Automatic Speech Recognition (ASR) to develop a body gesture recognition system in 2015 [20]. An HMM model was heavily used in gesture recognition system due to its stable performance. Nevertheless, recent studies in speech recognition showed a room for improvement in HMM. This study used artificial neural networks (ANNs) to estimate emission probabilities of HMMs. The usual practice was to use Gaussian Mixture Models (GMMs) for that purpose. The study used Microsoft Research Cambridge 12 (MSRC-12) which is a skeleton based dataset that contains 12 actions performed by 30 subjects. Experiments indicated that the proposed method achieved very competitive results.

Table 3.2: Summary of studies reviewed in Section 3.2

Author	Year	Application	Data	Features	Machine learning Technique
Chang	*2013*	Action recognition system to control PowerPoint	Online training and testing	Upper body joints' positions, and directional velocity of the joints.	SVM for online segmentation, and HMM to recognize the three speaker gestures.
Saha et al.	*2014*	Emotion recognition from body gestures	10 subjects performed 5 emotional gestures.	Hands distance with respect to spine, and angles between joints.	Decision Tree.
Arici et al.	*2014*	Human action recognition	38 subjects performed 12 different gestures (6 samples per gesture).	Joints 3D positions.	weighted DTW.
Ibanez et al.	*2014*	Human action recognition	4 subjects performed 7 different (20 samples per gesture)	Joints 3D positions.	DTW and HMM.
Wang et al.	*2013*	Two hand gestures recognition	8 two-hand gestures performed twice by 10 subjects.	Collection of six-vectors representing the world positions of the hands.	Coupled HMM.
Sharaf et al.	*2015*	Real time system for action detection	30 subjects performed 12 actions.	Joint angles and angular velocities derived from 3D joint world positions.	Linear SVM.
Gowayyed et al.	*2013*	Human action recognition	10 subjects performed 20 actions (2 or 3 times each).	Histogram of Oriented Displacements (HOD) of 2D joints trajectories.	Linear SVM.
Acorn et al.	*2015*	Anomaly sit-to-stand movement recognition	31 subjects performed sit-to-stand movement 6 times each. In addition to 24 anomaly samples.	Features extracted from DTW.	RF and k-NN.
Le et al.	*2015*	Body gesture recognition	30 subjects performed 12 actions.	Joints 3D positions.	HMM with Neural Networks for emission probabilities estimations.
Ding et al.	*2015*	Human gesture recognition	14 human gestures classes performed by 5 male subjects.	Eigenspace constructed out of joints 3D positions.	Nearest Neighbor classification rule.
Aujeszky et al.	*2015*	Body gesture recognition for Arabic sign language	6 Arabic signs performed by 10 subjects.	Joints 3D positions.	Nearest Neighbor classification rule.

3.3 Human Action Recognition using RGB-D and Body Joints Positions Features

Here, we inspect the feasibility of combining RGB-D streams and body joints tracking stream to recognize human gestures.

A recognition system that combined depth and body joints features was proposed by Keceli and Can in 2014 [75]. The system used a feature vector where the features can be classified into the following categories:

- **Joint Angles:** after evaluating all the angles between all joints in the skeleton, the study found out that the angles of shoulder-elbow-arm and hip-knee-foot are the most descriptive angles. The directed angle is calculated for the joints in all frames, then it was represented in the feature vector using histogram with 10 bins.

- **Displacements of Joints:** the study claimed that joints angles might not be efficient in some gestures like checking watch and crossing arms. Therefore, joint displacement was included in the feature vector. The joint displacement was calculated by aggregating all Euclidean distances from the consecutive frames. It is worth mentioning that the joint displacement is calculated for each dimension separately to get maximum knowledge about the movement.

- **Gradient Image:** a gradient of an image is a change intensity or color of that image. The study calculated the gradients of depth images on x, y,

Table 3.3: Summary of studies reviewed in Section 3.3

Author	Year	Application	Data	Features	Machine learning Technique
Keceli and Can	*2014*	Human action recognition	10 subjects performed 20 actions.	Joint Angles, Displacements of Joints, Gradient Image, and Depth History Image.	RF
Wu et al.	*2014*	Gesture-based authentication system	40 subjects (27 men and 13 women) where each subject performed 2 gestures.	Shape features for silhouettes, and joints positions for skeletal.	Rule-based

and z dimensions. Gradient image value is a summation of the commutative gradients in all dimensions.

- **Depth History Image:** the pixel intensity is a function of both the frequency and depth information which is known as depth history images (DHI). An important feature of DHI is that it stores what frames are more recent.

Wu et al. performed a comparison (in 2014) between silhouette features and skeletal features in a gesture-based authentication system [76]. The data was collected using 40 subjects (27 men and 13 women) where each subject performed 2 short gestures. Gestures were recorded under different circumstances (e.g. holding a bag and thick jackets) in order to test the robustness of the authentication system. The study concludes that the skeletal features were more robust than silhouette features.

3.4 Computer Graphics and Computer Vision

3D computer graphics is an important aspect of computer science. It is used to model, animate, and render 3D objects. In 3D graphics, a developer can model

human characters that have the same characteristics as real human body and behave in a similar manner. Synthesizing human data using 3D graphics was used by multiple studies in computer vision. The objective is to train the learning algorithm on comprehensive and diverse synthetic data to be sophisticated enough to tackle real human data.

Jalal et al. developed (in 2013) a human activity recognition system by recognizing the body parts of the human silhouette [77]. The proposed system was trained using synthetic depth images rendered using human models in 3Ds Max. The silhouettes of the synthetic depth images have 23 body parts labels. The descriptors of the synthetic data were the depth intensity differences of two pixels. Then, a random forest (RF) classifier [36] was trained on the synthetic data. To test the train model, a depth camera was used to collect 6 human activities (walking, running, right hand waving, lying down, sitting down and standing up) from 3 human subjects. The experimental findings indicate a superior performance by the proposed system.

Pose estimation is one of the key issues in computer vision. Synthetic data was employed to provide adequate training that can enhance the pose estimation accuracy of the learning algorithm. Kanaujia et al. proposed (in 2013) a new algorithm to estimate the human pose [78]. The proposed algorithm uses 3D visual hull which is constructed from multi-view silhouettes. The authors used motion capture files from HumanEva to smoothly deform human characters in AutoDesk Maya and render 2D image for pose estimation test. There were two

tests: part segmentation accuracy and joint localization accuracy. When using only the synthetic data for training the proposed algorithm showed a promising results for both tests on real data.

Another study that tackled pose estimation in 2015, Jeon et al. introduced unsupervised 3D human pose estimation [79]. The proposed method utilizes the geodesic distance map (GDM) of human depth silhouettes. The GDM was generated from the body centroid to all the pixels in the silhouette in order to indicate the distance between the pixel and the body centroid. Furthermore, the shortest paths were drawn using the Dijkstra algorithm from the body centroid towards the body extremities. The authors used 3Ds Max to simulate human poses, and the proposed algorithm estimated the skeleton poses efficiently.

Hand gesture recognition emerged to be an important topic in computer vision. Various studies in hand gesture recognition utilized 3D computer graphics to generate synthetic data. Dinh et al. designed a hand gesture recognition system for smart home appliances in 2014 [24]. The developers used 3Ds Max to generate synthetic dataset of hand silhouettes with hand parts labels. A Random Forests (RFs) classifier was trained using thousands of synthetic depth maps to identify the hand parts. Afterwards, the hand gesture was recognized based on the fingers states detected in the first stage. The experiments showed the efficiency of the proposed method.

A very similar approach of generating synthetic hand silhouettes can be found in [66] by Dinh et al. in 2014. The study illustrated the feasibility of using synthetic

data to train the learning algorithm in order to recognize hand number gesture.

Luong et al. developed a Human Computer Interaction (HCI) system based on hand gesture recognition [80]. The proposed system employed a random forest classifier that was trained on vast amount of synthetic images to recognize fingers and hand parts. The study used 3Ds Max to generate a synthetic dataset of hand silhouettes (800 image pairs). The extracted features were the depth value difference between pixels. The authors introduced an HCI system to control a mouse cursor in windows operating system, they selected two tasks: moving the cursor and double click.

In 2013, Lim et al. developed a pose estimation of upper human body [81]. The proposed method utilized the random forest classifier which is trained on synthetic images to recognize upper body parts. Then, support vector machine (SVM) classifier used to recognize the pose of upper body. The study used 3Ds Max to generate a synthetic dataset of upper body images (350 synthetic depth and labeled images). Moreover, the study collected 200 depth images from 10 subjects (100 for training and 100 for testing). The features used by SVM are angles between upper body joints. The experimental findings indicates high recognition rates by the proposed method.

Table 3.4: Summary of studies reviewed in Section 3.4

Author	Year	Application	Synthatic Data	Real Data	Features	Machine learning Technique
Jalal et al.	*2013*	Human activity recognition	Synthetic depth images (labelled with 23 body parts) rendered using human models in 3Ds Max.	3 subjects performed 6 actions.	Depth intensity differences of two pixels.	RF
Kanaujia et al.	*2013*	Human pose estimation	MoCaps from HumanEva imported into AutoDesk Maya and rendered as 2D images.	3 subjects performed 2 actions.	3D visual hull which is constructed from multi-view silhouettes.	SVM and RF
Jeon et al.	*2015*	Human pose recognition	50 poses generated using 3Ds Max.	2 subjects performed free poses.	Geodesic distance map (GDM) of human depth silhouettes.	k-NN
Dinh et al.	*2014*	Hand gesture recognition	3Ds Max used to generate silhouettes of the hand with 12 hand parts labels.	400 samples representing four hand gestures from five subjects.	Hand depth silhouette to recognize hand parts. Recognized hand parts were used to identify the gesture.	Hand parts recognition using RFs. Recognition rules to determine the gesture.
Dinh et al.	*2014*	Hand number gesture recognition	3Ds Max used to generate silhouettes of the hand (5,000 pairs) with 12 hand parts labels.	500 samples representing 10 hand number gestures from five subjects (100 each).	Hand depth silhouette to recognize hand parts. Recognized hand parts were used to identify the gesture.	Hand parts recognition using RFs, recognition rules to determine the number gesture.
Luong et al.	*2013*	Hand gesture recognition	3Ds Max used to generate 800 silhouettes of the hand with hand parts labels.	Hand silhouettes captured by depth camera.	Depth intensity differences of two pixels.	RF
Lim et al.	*2013*	Human pose estimation	3Ds Max used to generate 350 silhouettes of the upper human body.	200 depth images collected by 10 subjects.	Depth intensity differences of two pixels and joints angle features.	SVM and RF

CHAPTER 4

PROPOSED APPROACH

The considerable amount of studies in literature that addressed the human action recognition indicates the importance of this problem. Moreover, recent computer vision technologies (such as body joints tracking system [18]) enable researchers to build learning algorithms on top of invariant and robust systems. In this thesis we tackled the problem of human action recognition by utilizing body joints tracking system [18] provided by Kinect.

4.1 Problem Statement

In this thesis, we mainly address two issues that continue to persist in human action recognition research: *Extracted Features* and *Training Data*. Despite the vast amount of studies that addressed the human action recognition problem, the feature extraction part still needs more work in order to propose more meaningful and discernment features. Some studies used the raw positions of joints as features; however, the subjects body built can easily confuse this method or his behavior

while performing the action.

For the training data, most of the studies collected their training data using Kinect. Nevertheless, this process can be time consuming if we consider the communication time with the subject and the time consumed to explain the action for the subject. Additionally, noisy samples can occur during the recording session, which adds overhead to the process. we selected the study published by Ibañez et al., 2014 [2] as our baseline in terms of selected joints and action data structure.

4.2 Research Hypotheses

Based on the problem statement discussion we formulate our hypothesis structure as follows:

- H1: **(The influence of extracted features)**, extracting features from joints positions can produce better recognition rates than using raw joints positions as features.

  (Null hypothesis: extracting features from joints positions cannot produce better recognition rates than using raw joints positions as features).

- H2: **(Training the model using synthetic data)**, synthetic data can provide comprehensive training for the learning algorithm.

  (Null hypothesis: synthetic data cannot provide a comprehensive training for the learning algorithm).

Table 4.1: Summary of Experimental Designs Carried out in this Thesis

	Objective	Actions	Joints	Training Data	Testing Data	Preprocessing	Extracted Features
Experimental Design 1	Improve the recognition rates of EasyGR against novel data.	• Circle • Elongation • Swim • Smash • Punch • Swipe Left • Swipe Right	• Head • Neck • SpineMid • ShoulderLeft • ElbowLeft • HandLeft • ShoulderRight • ElbowRight • HandRight • HipLeft • KneeLeft • FootLeft • HipRight • KneeRight • FootRight.	560 samples (80 samples per action) recorded using Kinect by 4 subjects from EasyGR research team	140 samples (20 samples per action) recorded using Kinect by 2 subjects from our research team	• Translation • Normalization	• Standard deviation of joints positions. • 3D and maximum1D Euclidean distance between: - First and middle frames. - First and last frames. • 8 statistical properties of Discrete Wavelet Transform (DWT) coefficients performed on body extremities.
Experimental Design 2	Inspect the feasibility of using synthetic data to provide a comprehensive training for the learning algorithm.	• Balance • Jump • Kick • Pickup • Punch • Stretch	• Head • Neck • SpineMid • ShoulderLeft • ElbowLeft • HandLeft • ShoulderRight • ElbowRight • HandRight • HipLeft • KneeLeft • FootLeft • HipRight • KneeRight • FootRight.	972 samples (162 samples per action) generated using 3D graphics, performed by 6 human characters	120 samples (20 samples per action) recorded using Kinect by 2 subjects from our research team	• Translation • Normalization • Quantization	• Standard deviation of joints positions. • 3D and maximum 1D Euclidean distance between: - First and middle frames. - First and last frames. • 8 statistical properties of Discrete Wavelet Transform (DWT) coefficients performed on body extremities.

Therefore, we propose two experimental approaches in order to address each hypothesis independently. Table 4.1 shows the parameters of both experimental designs (e.g., joints used, dataset size, and extracted features). Both experimental designs differ by their objectives. In the first experimental design, we searched for new features to describe the human action, while in the second experimental design we investigated the feasibility of using synthetic data for training. Next, we discuss the human action data used in this work, the preprocessing techniques used, and nature of the extracted features.

4.3 Human Action Data

The authors of (EasyGR [2]) generously shared their data with us. The data of EasyGR [2] is stored in text files where each text file represent an action. Therefore, the text files are the data samples (also known as data instances). Each text file contains a matrix of size $N \times 3M$ where:

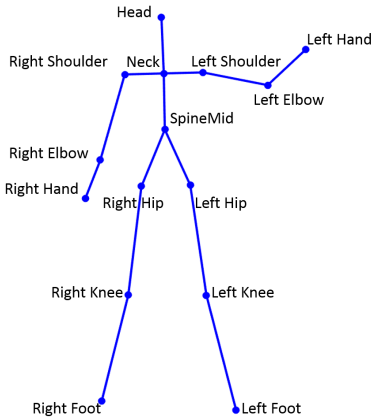


Figure 4.1: The human joints tracked in this thesis

$$
\begin{array}{c}
 \\
t_1 \\
t_2 \\
 \\
t_N
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc}
joint_1 & & & joint_2 & & & \dots & joint_M & &
\end{array} \\
\left(\begin{array}{cccccccccc}
x_{11} & y_{11} & z_{11} & x_{12} & y_{12} & z_{12} & \dots & x_{1M} & y_{1M} & z_{1M} \\
x_{21} & y_{21} & z_{21} & x_{22} & y_{22} & z_{22} & \dots & x_{2M} & y_{2M} & z_{2M} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_{N1} & y_{N1} & z_{N1} & x_{N2} & y_{N2} & z_{N2} & \dots & x_{NM} & y_{NM} & z_{NM}
\end{array}\right)
\end{array}
$$

- N is the number of recorded frames (Assuming t is a single unit of time which is the frame number such that $t \in [1..N]$). N was set to be adequate to capture the human action.

- M is the number of tracked joints (in this study $M = 15$ see Figure 4.1) multiplied by 3 to represent the three dimensional space: x (horizontal axis), y (vertical axis), and z (depth value).

4.3.1 Experimental Design 1

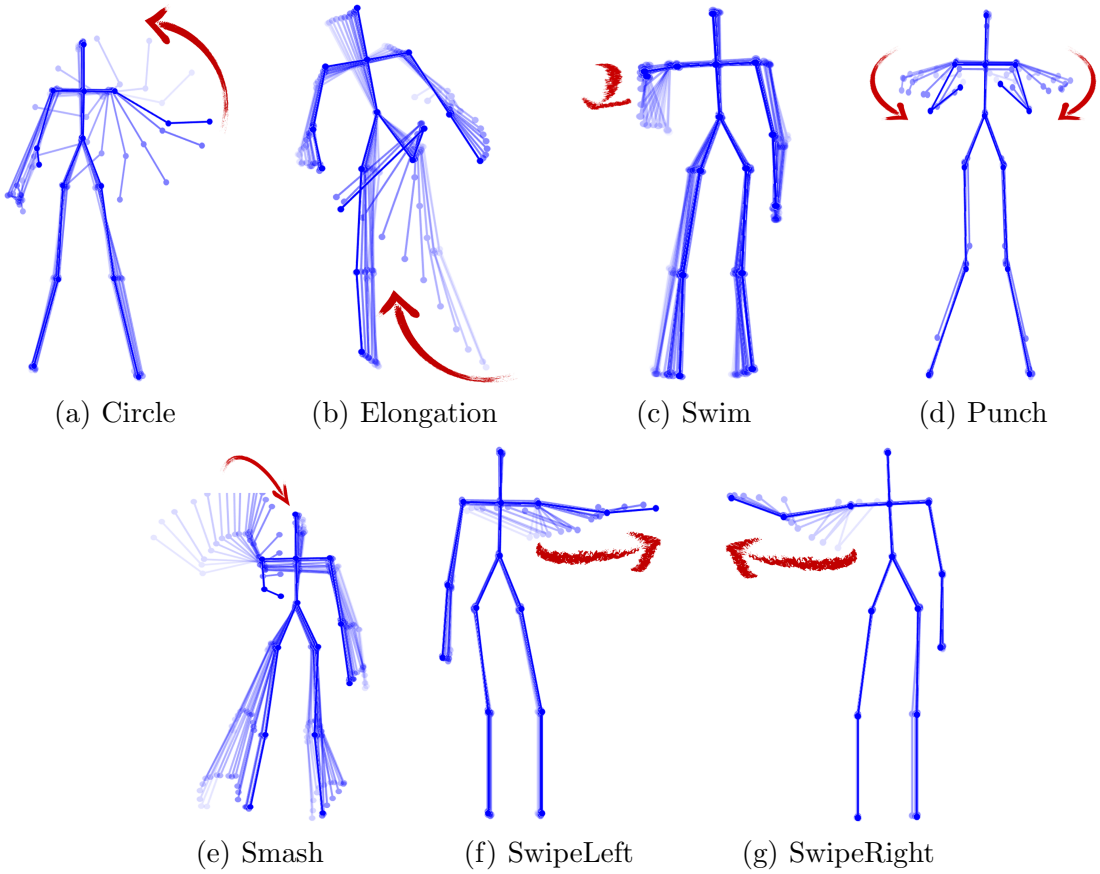
In this experimental design we selected EasyGR [2] to be our baseline study. In EasyGR, authors used the plain coordinates of the joints as features that used by the DTW and HMM to recognize the human action. Although the plain coordinates were preprocessed, they are still not robust enough against variations in subjects body built and action behavior. Our objective in this experimental design is to search, investigate, and propose influential features to describe the human action using the same data in EasyGR [2].

The EasyGR dataset contains seven actions illustrated in Figure 4.2: Circle, Elongation, Swim, Smash, Punch, Swipe Left, and Swipe Right. Two categories of data used in this experimental design:

- **Training Data:** 560 samples (80 samples per action), performed by four subjects from EasyGR research team with different body builds.
- **Testing Data:** 140 samples (20 samples per action), performed by two subjects from our research team with different body builds.

4.3.2 Experimental Design 2

In this experimental design, we aim to investigate the feasibility of training the classifier on synthetic human action data. Recording real human action data is an intensive task since it requires some introductory steps (e.g. communicating with the subject and explain to him/her how to perform the action). Computer



(a) Circle (b) Elongation (c) Swim (d) Punch

(e) Smash (f) SwipeLeft (g) SwipeRight

Figure 4.2: Actions in EasyGR dataset [2]

3D graphics software are sophisticated enough to simulate human actions without the need of introductory steps.

The human action data in this experimental design contains six actions: Balance, Jump, Kick, Pickup, Punch, and Stretch. Two categories of data used in this experimental design:

- **Training Data:** 972 samples (162 samples per action), performed by six synthetic subjects with different body builds designed using AutoDesk Character Generator software.

- **Testing Data:** 120 samples (20 samples per action), performed by two

subjects from our research team with different body builds.

Next, we explain the process of generating both datasets.

Training Data

To generate human action synthetic data we used AutoDesk Maya 2015. Maya is a 3D graphics software with innovative HumanIK (Human Inverse Kinematics) tool, which provides a natural deformation of human characters. To simulate a human action, we need two components:

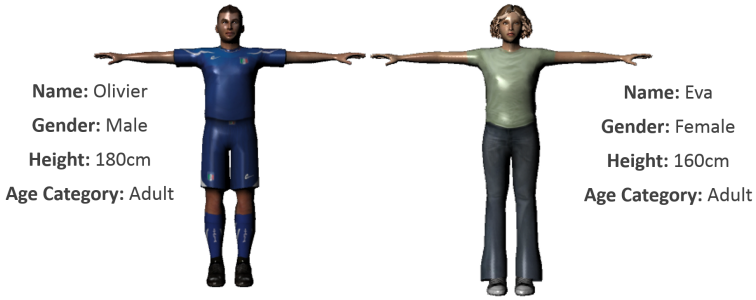
- **Rigged human character :** is a combination of a human mesh and set of joints that mimic the human skeleton. We generated the rigged human models using AutoDesk Character Generator [82].

- **Motion capture (MoCap) file :** the file that contains the motion information. We downloaded free MoCaps from NUS (National University of Singapore) Motion Capture Database [83].

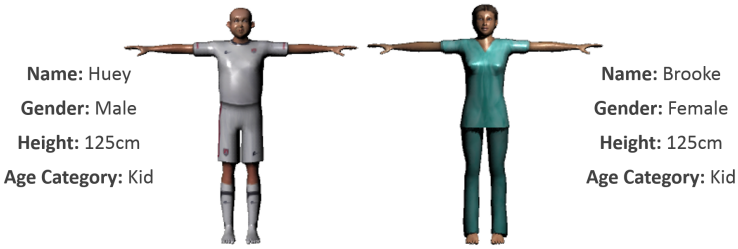
AutoDesk character generator was used to generate six human subjects. During the design of human characters we consider the diversity in: gender, body build, age category. The specifications of human characters are illustrated in Figure 4.3.

Afterwards, we imported the human characters one by one into a Maya scene. Then, apply motion information files (MoCaps) onto these characters. The gesture action was then captured from different perspectives using 27 cameras positioned at specific distances and angles from the human character spine in the virtual

(a) Human Characters with Large Body Build



(b) Human Characters with Average Body Build



(c) Human Characters with Small Body Build

Figure 4.3: Human Characters used in Experimental Design 2

3D environment. The distribution of cameras is shown in Figure 4.4. Moreover, Table 4.2 illustrates the cameras positions according to the spherical coordinates around the human character spine.

For a single action we generated 162 samples, since we have 6 human subjects and 27 cameras positioned to capture the action from different angles. As a result, we obtained 972 human action samples for all of the 6 actions, which is the size of the training data.

Figure 4.4: Setup of generated cameras

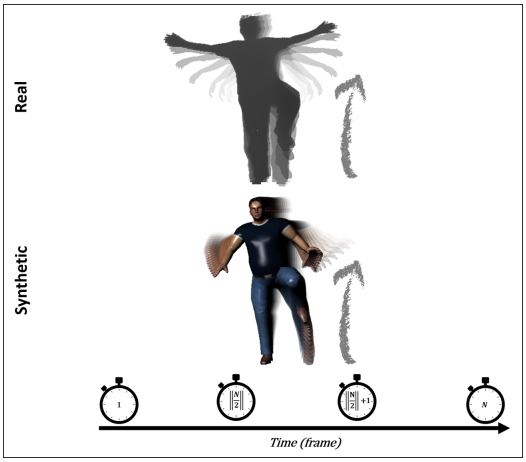
Table 4.2: Cameras Positions in Maya Scene

	r	θ	φ		r	θ	φ		r	θ	φ
Camera 1	300	0°	0°	**Camera 10**	300	90°	30°	**Camera 19**	300	90°	-30°
Camera 2	300	0°	10°	**Camera 11**	300	70°	30°	**Camera 20**	300	70°	-30°
Camera 3	300	0°	20°	**Camera 12**	300	50°	30°	**Camera 21**	300	50°	-30°
Camera 4	300	0°	30°	**Camera 13**	300	30°	30°	**Camera 22**	300	30°	-30°
Camera 5	300	0°	40°	**Camera 14**	300	10°	30°	**Camera 23**	300	10°	-30°
Camera 6	300	0°	-10°	**Camera 15**	300	110°	30°	**Camera 24**	300	110°	-30°
Camera 7	300	0°	-20°	**Camera 16**	300	130°	30°	**Camera 25**	300	130°	-30°
Camera 8	300	0°	-30°	**Camera 17**	300	150°	30°	**Camera 26**	300	150°	-30°
Camera 9	300	0°	-40°	**Camera 18**	300	170°	30°	**Camera 27**	300	170°	-30°

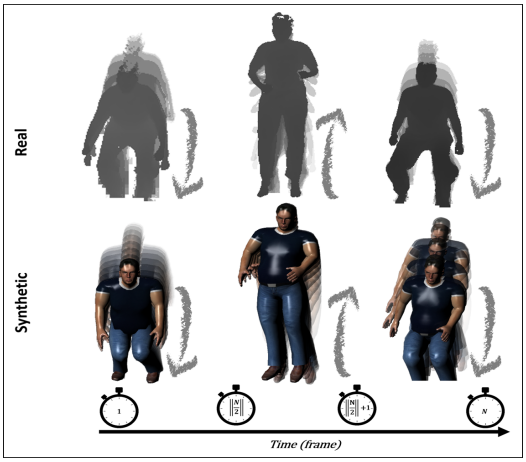
Testing Data

Kinect v2 was used to record real human actions performed by two subjects from our research team. The Kinect was installed on a table with approximately 3 meters distance from the subject. The recording performed in ordinary living room.

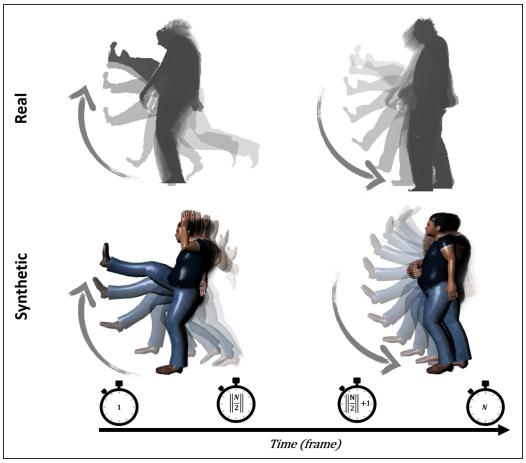
The 15 joints (refer to Figure 4.1) world positions as well as depth images (with instant background subtraction) were recorded and stored during the actions. The depth images used to segment the file of joints world positions into samples since all the trials performed at once. Figure 4.5 shows an overview about actions, synthetic data, and real data used in this experimental design.



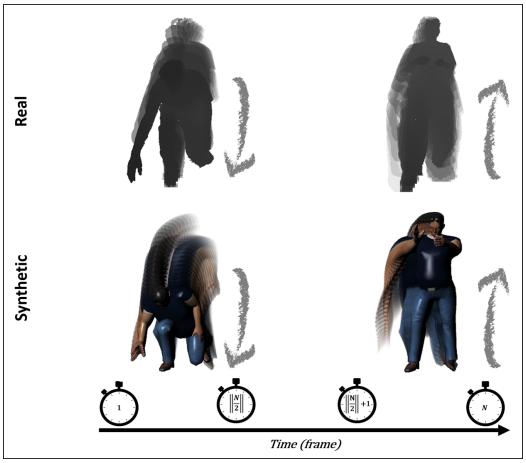
(a) Balance



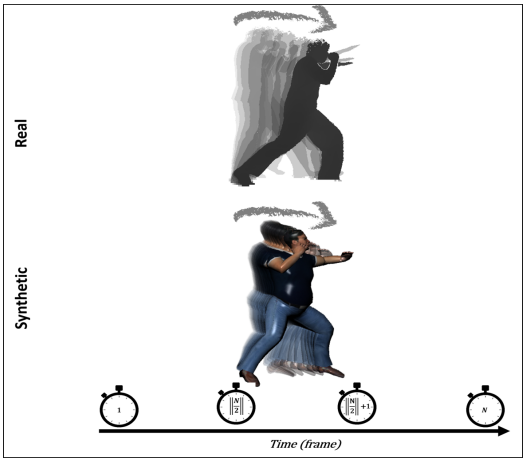
(b) Jump



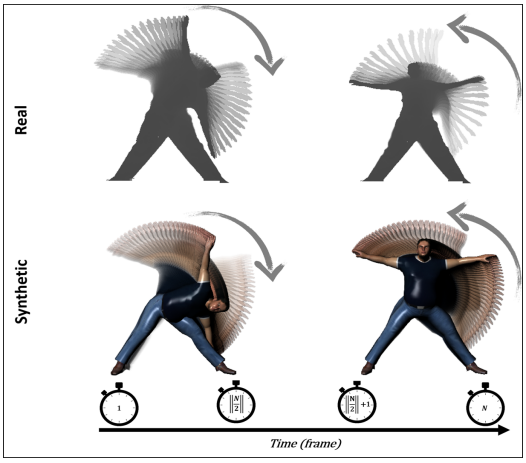
(c) Kick



(d) Pickup



(e) Punch



(f) Stretch

Figure 4.5: Actions in Experimental Design 2

4.4 Preprocessing

When recording an action using Kinect, the subject's position and his size variations need to be considered before performing the training or testing. Translation and normalization are two preprocessing steps used to overcome aforementioned variations [2, 19].

In our second experimental design, we introduced a new preprocessing step, which is quantization to simplify the classification model. In the upcoming subsections, we discussed these preprocessing techniques in details.

4.4.1 Translation

Translation preprocessing illustrated in Figure 4.6, used to overcome the subjects translation variations. The centroid of joint positions needs to be subtracted from all positions. Some studies used the position of the hip center as centroid. However, we adopted the method introduced by [2] to calculate the centroid as illustrated in Equation 4.1, where n is the total number of joints.

$$centroid = (\bar{x}, \bar{y}, \bar{z}) = \frac{\sum_{i=1}^{n}(x_i, y_i, z_i)}{n} \tag{4.1}$$

Then, the centroid of the positions was subtracted from all positions as shown in Equation 4.2.

$$(x_i, y_i, z_i) = (x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z}) \tag{4.2}$$

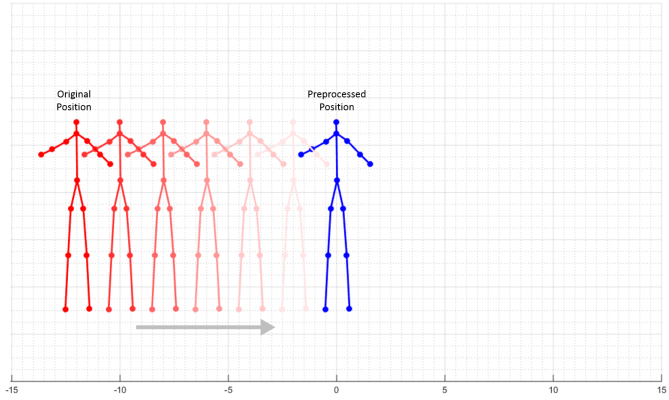


Figure 4.6: Translation Preprocessing

4.4.2 Normalization

Normalization preprocessing shown in Figure 4.7, used to overcome the variations in different body builds. The joints positions were normalized by the distance between the neck and the spine (called SpineMid in Kinect v2). Equation 4.3 demonstrates the formula to find the Euclidean distance between the neck and the spine. Furthermore, Equation 4.4 shows the normalization of a single joint position.

$$d = \sqrt{(x_{Neck} - x_{Spine})^2 + (y_{Neck} - y_{Spine})^2 + (z_{Neck} - z_{Spine})^2} \tag{4.3}$$

$$(x_i, y_i, z_i) = \frac{(x_i, y_i, z_i)}{d} \tag{4.4}$$

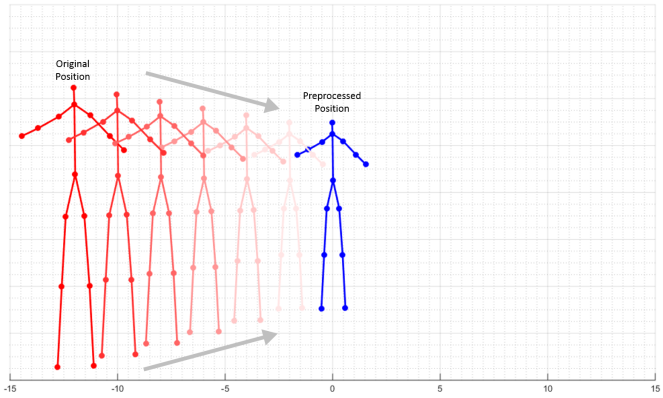
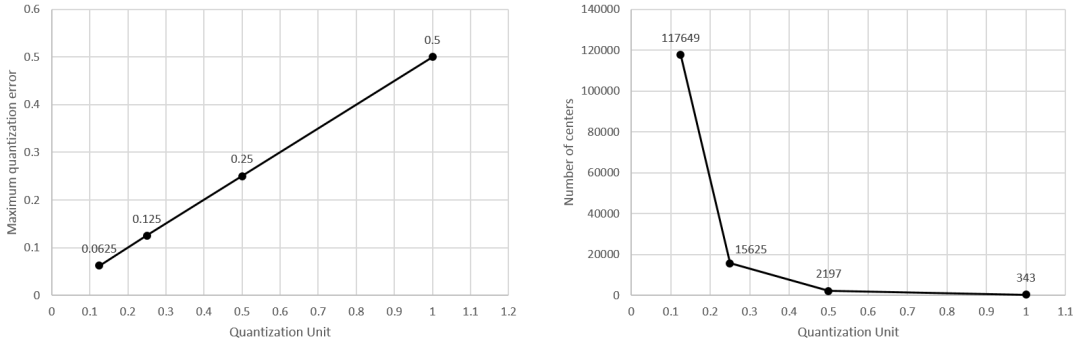


Figure 4.7: Translation and Normalization Preprocessing

4.4.3 Quantization

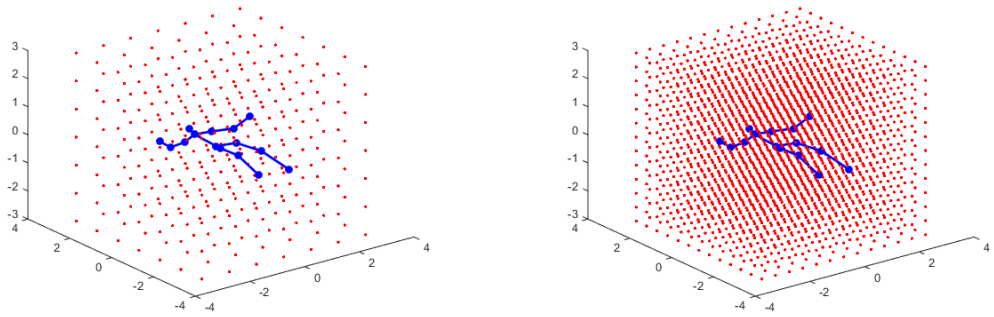
Quantization (or more specifically Vector Quantization) is the process of assigning set of points in the space to a finite number of groups, where each group is represented by its center [84]. There are machine learning techniques that adopted the principle of vector quantization (e.g. learning vector quantization [85] and k-means clustering).

The initial assumption constitute that performing quantization ahead of the classification process could enhance the learning process, because the observations are limited into predefined groups. The first intuition is the distance between the centers is a crucial parameter. If the distance between the centers is too small the quantization step would not help much because of large number of groups. On the other hand, if the distance between the centers is too large, it may lead to information loss because the observations moved significantly from their original locations. Therefore, the distance between the centers has to be set carefully as can be seen later in the text.



(a) Maximum Quantization Error (MQE) (b) Quantization Centers

Figure 4.8: Evaluation of four quantization schemes



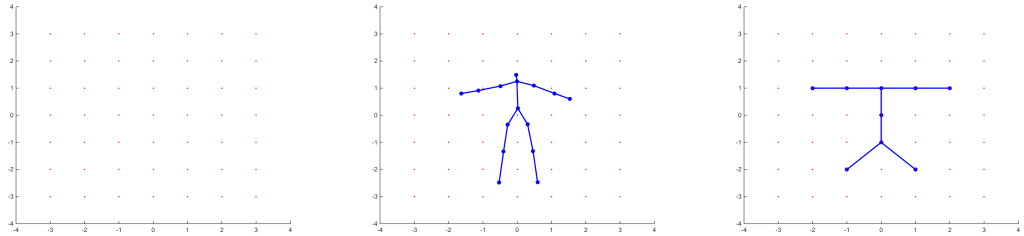
(a) One Unit (b) Half Unit

Figure 4.9: Quantization Schemes in Experimental Design 2

Before implementing quantization, we evaluated 4 quantization schemes: 1, 0.5, 0.25, and 0.125 units. It is important to note that a single unit in the Kinect coordinates system equals to one meter. We evaluated aforementioned schemes using: Maximum Quantization Error (MQE) (illustrated in Equation 4.5) and number of generated centers.

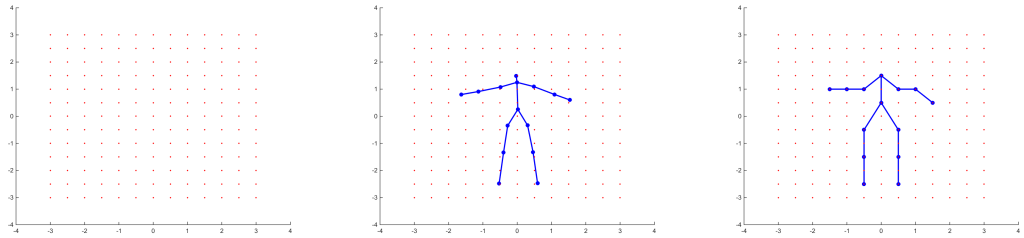
$$MQE = \frac{\Delta}{2} \quad \text{, where } \Delta = \text{quantization step} \tag{4.5}$$

Based on the observations provided by Figure 4.8, we selected one unit and



(a) Quantization Centers (b) Non Quantized Skeleton (c) Quantized Skeleton

Figure 4.10: Quantization preprocessing using One Unit



(a) Quantization Centers (b) Non Quantized Skeleton (c) Quantized Skeleton

Figure 4.11: Quantization preprocessing using Half Unit

half unit quantization schemes. The former is the most intuitive scheme and it provides a baseline for other schemes, while the latter represents the best tradeoff between maximum quantization error and number of centers. Overview of both schemes provided in Figure 4.9.

Assigning a joint to its nearest quantization center performed according to the Euclidean distance. Furthermore, Figure 4.10 and Figure 4.11 illustrates how the quantization was performed on the joints positions obtained by Kinect v2.

4.5 Features Extraction

In both experimental designs, our feature extraction approach consists of three types of features:

- Standard deviation of all joints positions with respect to time on each axis separately.
- Max 1D and 3D Euclidean distances between the joint position in the first frame ($t = 1$) and the position in the middle frame ($t = ||\frac{N}{2}||$) in addition to the distance between its position in the first frame ($t = 1$) and the position in last frame ($t = N$).
- The statistical analysis of Discrete Wavelet Transform (DWT) signals performed on body extremities.

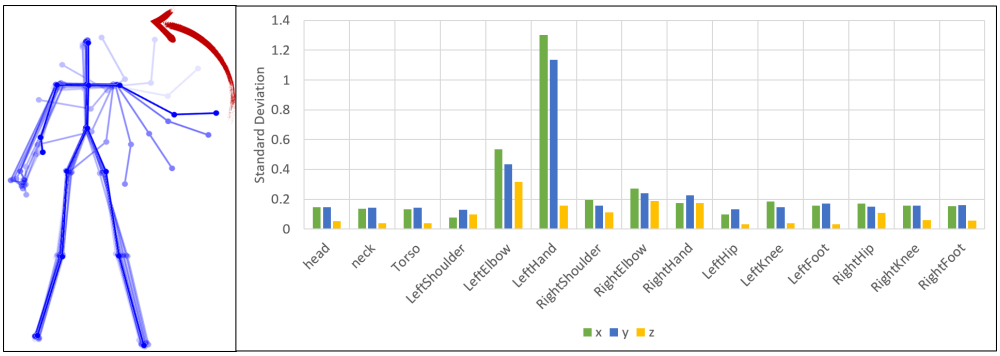
Next, we describe these features in details.

4.5.1 Standard Deviation

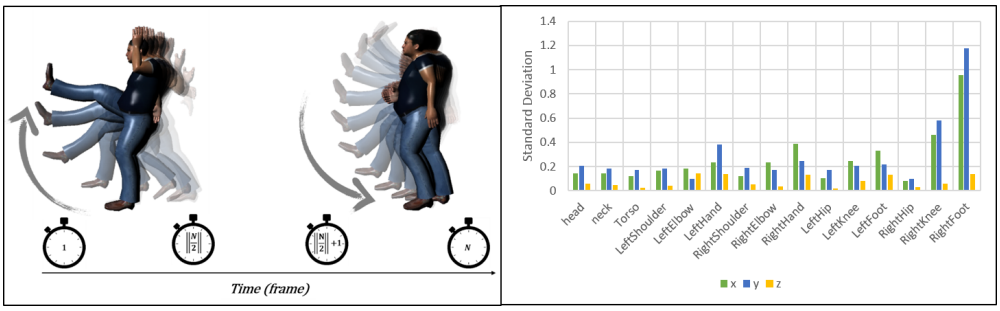
The standard deviation measures the amount of variation of a set of data samples, it is calculated using Equation 4.6, where N is the total number of data points, x is the current point, and μ is the mean of the data points.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{4.6}$$

The standard deviation of the joints positions with respect to time reveals important information about which joints moved during the action. This feature represents a vital descriptor to differentiate between actions. Figure 4.12(a) and Figure 4.12(b) illustrate how the standard deviation can be used to reveal the interest joint in an action. The size of this feature vector is 45, since we are calculating the standard deviation on each column of the text file.



(a) Circle action in experimental design 1



(b) Kick action in experimental design 2

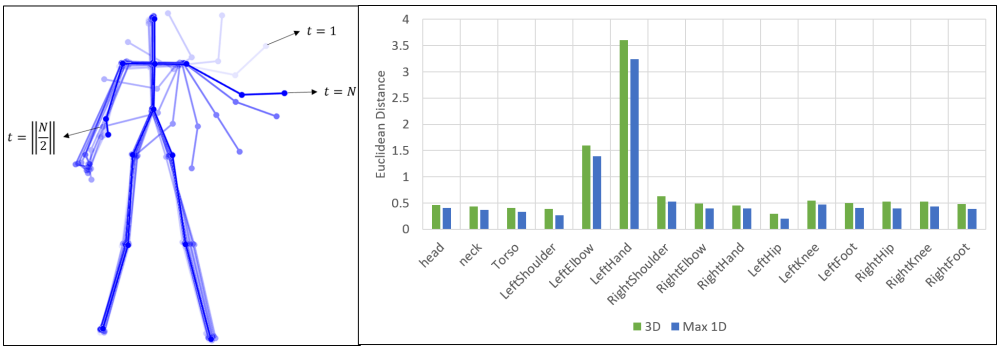
Figure 4.12: Discriminative ability of standard deviation features vector

4.5.2 Euclidean Distance

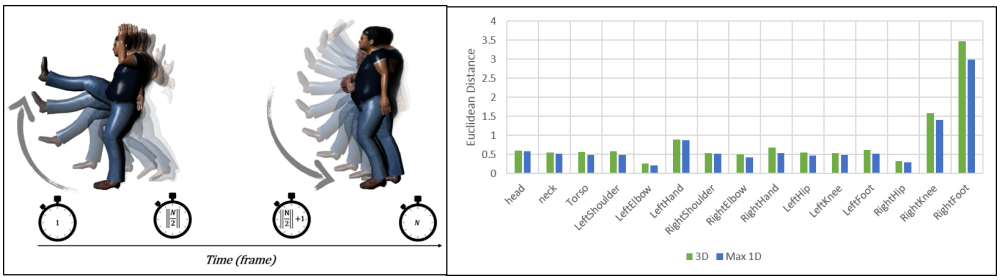
Measuring the Euclidean distance between two joint positions in different timings describes the nature of the joint movement. Here, we coupled the maximum 1D Euclidian distance with 3D Euclidian distance to get benefit of both and to enhance the discriminative ability. Using 1D space the Euclidean distance calculated by Equation 4.7 (where x_1 and x_2 are values in a single dimension). 3D Euclidean distance calculated using Equation 4.8.

$$d = \sqrt{(x_2 - x_1)^2} \tag{4.7}$$

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{4.8}$$



(a) Circle action in experimental design 1



(b) Kick action in experimental design 2

Figure 4.13: Discriminative ability of Euclidean distance features vector

Figure 4.13(a) and Figure 4.13(b) illustrate the discriminative ability of 1D Euclidean distance between first frame ($t = 1$) and the middle frame ($t = ||\frac{N}{2}||$). Two distances calculated for each joint:

- between the joint position in the first frame ($t = 1$) and its position in the middle frame ($t = ||\frac{N}{2}||$) we extracted the following:
  - Maximum of 1D Euclidean distances (on x-axis, y-axis, and z-axis)
  - 3D Euclidean distance.
- between the joint position in the first frame ($t = 1$) and its position in the last frame ($t = N$) we extracted the following:
  - Maximum of 1D Euclidean distances (on x-axis, y-axis, and z-axis)

– 3D Euclidean distance.

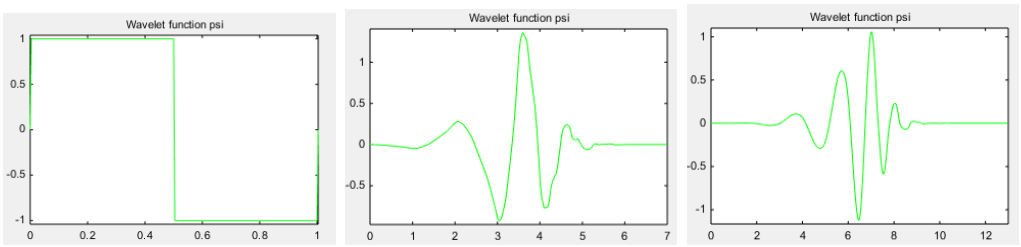
As a result, the size of this feature vector is 60, since we have fifteen 1D distances and fifteen 3D distances up to $t = ||\frac{N}{2}||$, in addition to another fifteen 1D distances and fifteen 3D distances up to $t = N$.

4.5.3 Statistical Properties of Discrete Wavelet Transform Signals

The DWT transform is a mathematical representation of signals and functions. DWT is used to perform deep analysis for numerical values because they inherit the same properties of the original signal over a finite interval. Moreover, Wavelet transform maintains the temporal relations between numerical values.

Analyzing the human joint trajectory using the DWT transform can reveal important information about the human action. In this study, we used Daubechies wavelet family to analyze the human joint trajectory. We selected three mother wavelets (db1, db4, and db7) shown in Figure 4.14. This selections was due to similarities between Daubechies mother wavelets, db1, db4, and db7 can represent the subgroups db1-db3, db4-db6, and db7-db10 respectively [86]. The DWT performed on one level of decomposition. The process of DWT illustrated in Figure 4.15, where the original signal passed through a low pass filter (g) to generate the approximation coefficients. Then it passed a high pass filter (h) to generate the detail coefficients.

To demonstrate the discriminative ability of DWT signals, we analyzed the



(a) db1 (b) db4 (c) db7

Figure 4.14: Illustration of the Daubechies wavelet

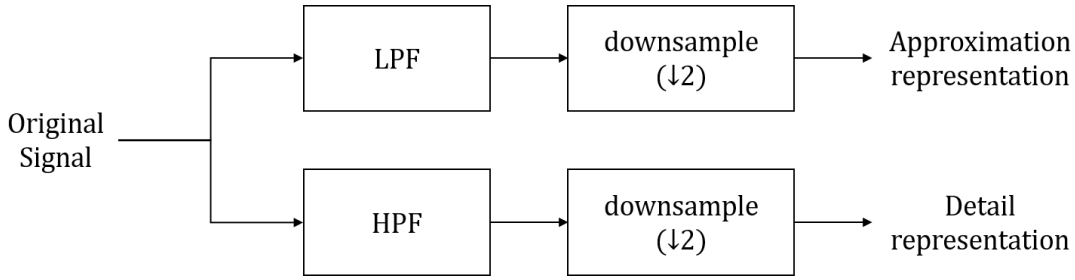
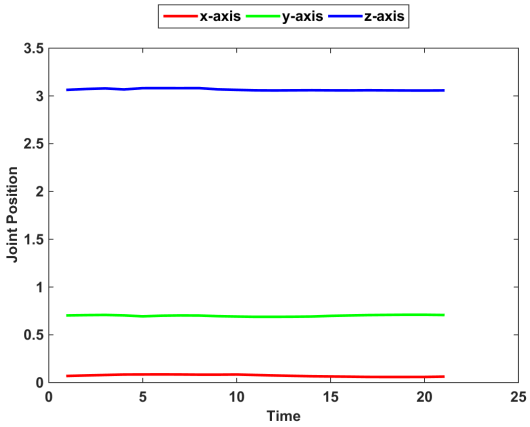
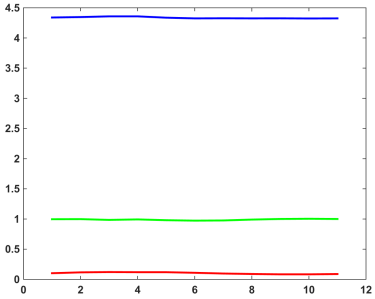


Figure 4.15: DWT Signal Decomposition

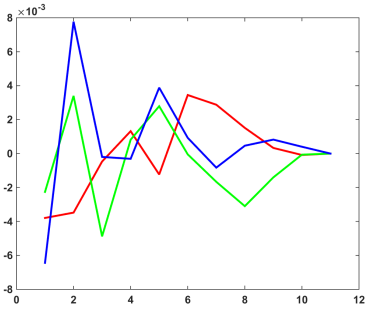
data captured by Kinect sensor of circle and kick actions. Figure 4.16 shows the analysis of the head joint during the circle action, which appears to be constant since it is not moving in this particular action. Nevertheless, Figure 4.17 shows the analysis of LeftHand joint (i.e., the interest joint in circle action) and it uncovers important information with high discrimination potential. Figure 4.18 and Figure 4.19 show the trajectories of head and right foot in kick action respectively. Similar to circle action, the head joint appears to be steady during the kick action. However, the trajectories of the right foot joint (i.e., the interest joint in kick action) contain a lot of information.



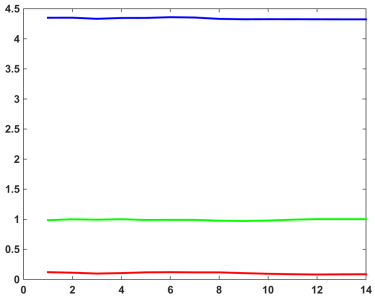
(a) Original Trajectory



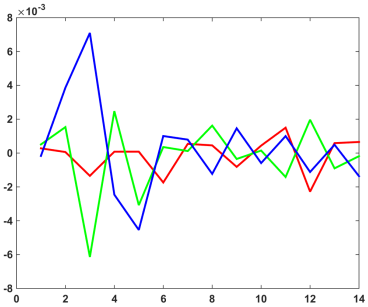
(b) db1 Approx. signal



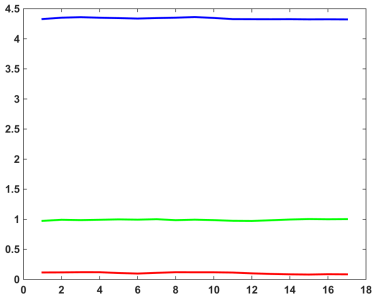
(c) db1 Detail signal



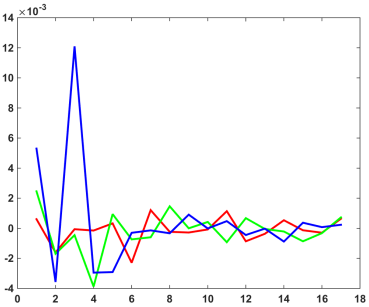
(d) db4 Approx. signal



(e) db4 Detail signal

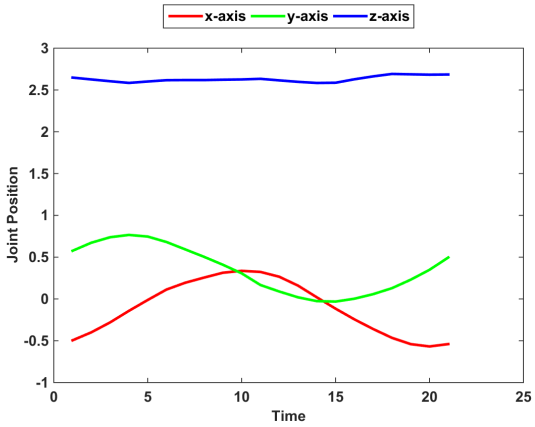


(f) db7 Approx. signal

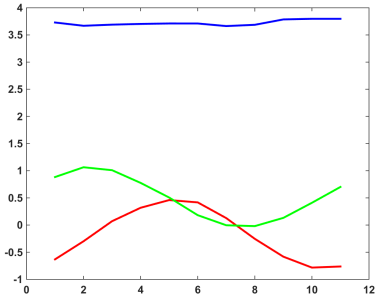


(g) db7 Detail signal

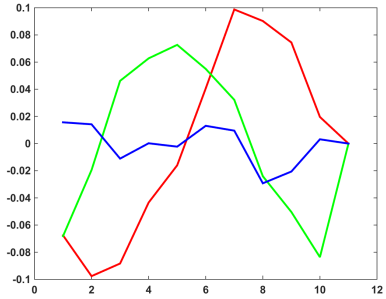
Figure 4.16: Head joint in a circle action of experimental design 1



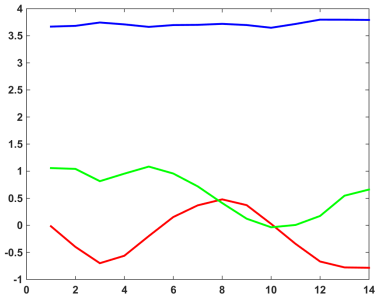
(a) Original Trajectory



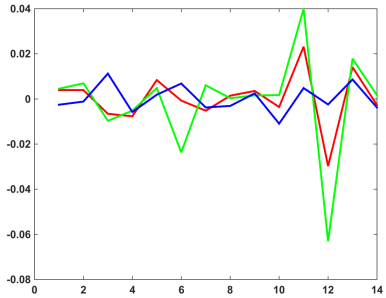
(b) db1 Approx. signal



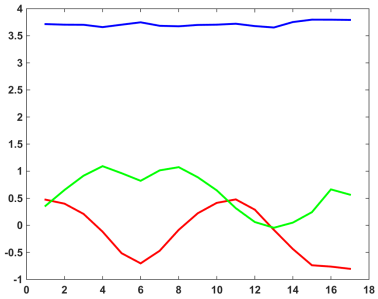
(c) db1 Detail signal



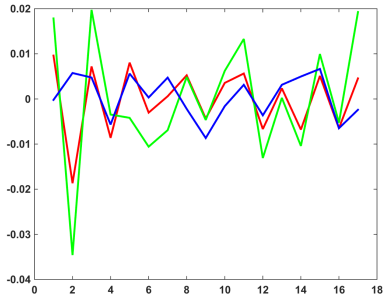
(d) db4 Approx. signal



(e) db4 Detail signal

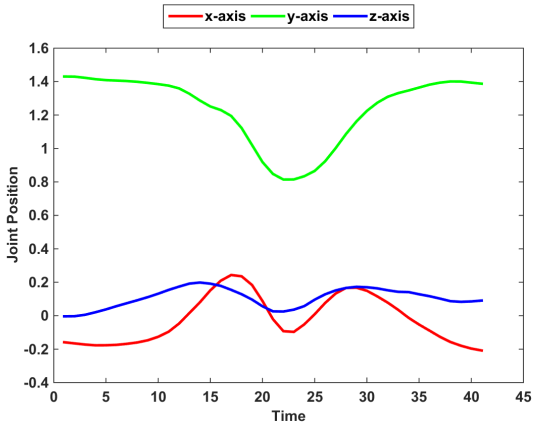


(f) db7 Approx. signal

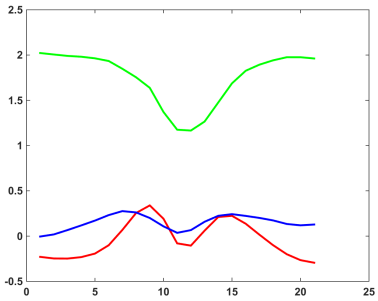


(g) db7 Detail signal

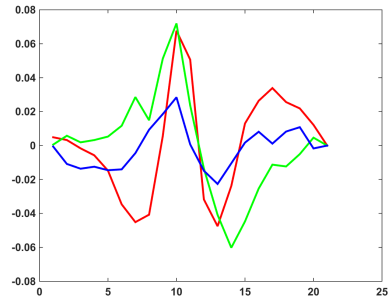
Figure 4.17: Left Hand joint in a circle action of experimental design 1



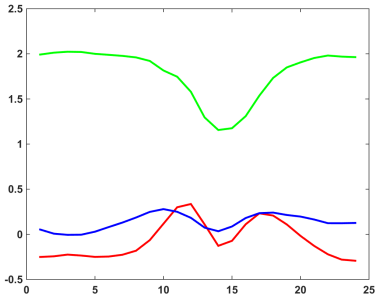
(a) Original Trajectory



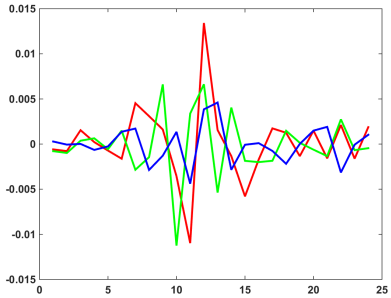
(b) db1 Approx. signal



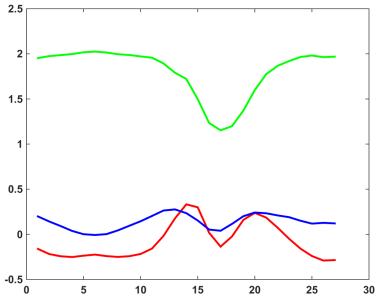
(c) db1 Detail signal



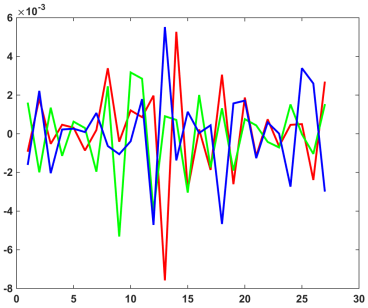
(d) db4 Approx. signal



(e) db4 Detail signal

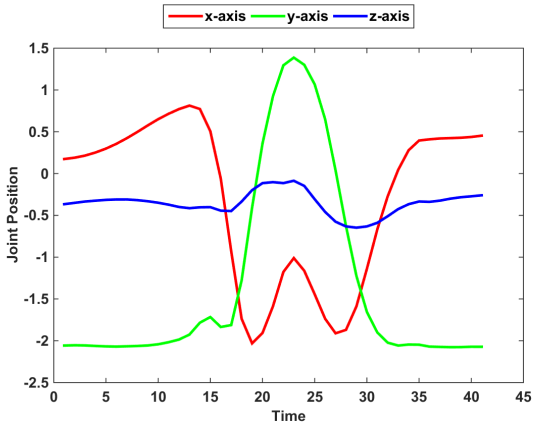


(f) db7 Approx. signal

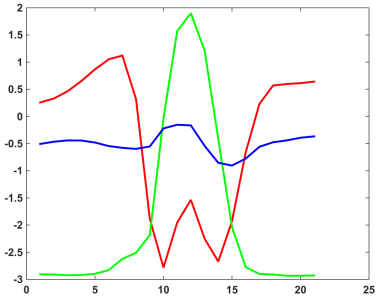


(g) db7 Detail signal

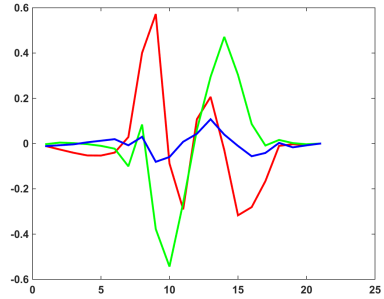
Figure 4.18: Head joint in a kick action of experimental design 2



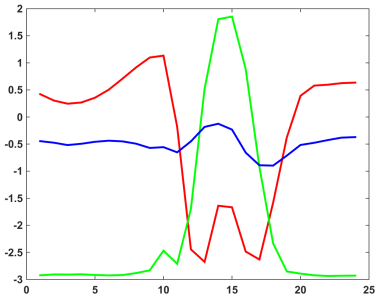
(a) Original Trajectory



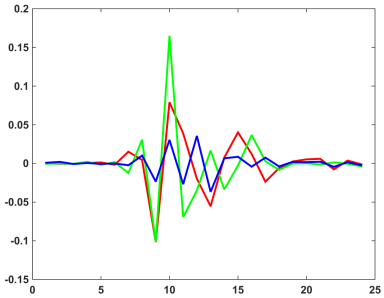
(b) db1 Approx. signal



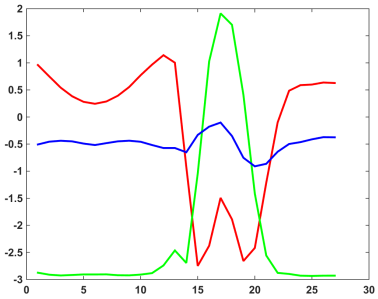
(c) db1 Detail signal



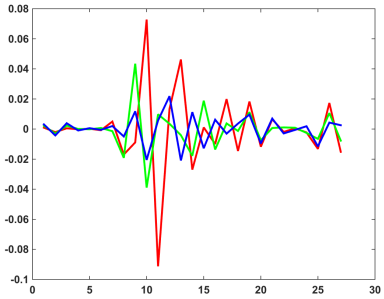
(d) db4 Approx. signal



(e) db4 Detail signal



(f) db7 Approx. signal



(g) db7 Detail signal

Figure 4.19: Right Foot joint in a kick action of experimental design 2

After analyzing all human joints trajectories, we found out that the human actions accumulate on the human body extremities. Therefore, the DWT applied only on the body extremities trajectories (i.e., Left Hand, Right Hand, Left Foot, and Right Foot). We extracted the following statistical properties of the approximation coefficients and the detail coefficients:

- Standard Deviation
- Maximum
- Minimum
- 1^{st} Quantile
- 2^{nd} Quantile (i.e., Mean)
- 3^{rd} Quantile
- Skewness
- Kurtosis

On one hand, we selected the first six statistical properties to keep the model simple. On the other hand, the latter duo statistical properties (i.e., skewness and kurtosis) chosen to capture the nonlinear behavior of the function. Two features vectors generated using the statistical properties of DWT. The first features vector contains DWT approximation coefficients and the second features vector contains DWT detail coefficients. The size of these features vector is 96, since we have:

- Four joints (i.e., Left Hand, Right Hand, Left Foot, and Right Foot).
- Eight statistical properties (i.e., Standard Deviation, Maximum, Minimum, 1st Quantile, 2nd Quantile, 3rd Quantile, Skewness, and Kurtosis).

Table 4.3: Features Vectors Constructed from Extracted Features

No.	Abbr.	Size	Used Joints	Description
1	Stdev (V1)	45	15	Standard divination of every joint in a single dimension.
2	Euc (V2)	60	15	Maximum 1D Euclidean distance and 3D Euclidean distance between the joint position in the first frame and its position in the middle and last frames.
3	DWTcAStat (V3)	96	4	8 Statistical properties of the DWT approximation coefficients, performed on body extremities trajectories (i.e. Left Hand, Right Hand, Left Foot, and Right Foot).
4	DWTcDStat (V4)	96	4	8 Statistical properties of the DWT detail coefficients, performed on body extremities trajectories (i.e. Left Hand, Right Hand, Left Foot, and Right Foot).
5	V1+V2	105	-	Feature vector #1 combined with Feature vector #2
6	V1+V3	141	-	Feature vector #1 combined with Feature vector #3
7	V1+V4	141	-	Feature vector #1 combined with Feature vector #4
8	V2+V3	156	-	Feature vector #2 combined with Feature vector #3
9	V2+V4	156	-	Feature vector #2 combined with Feature vector #4
10	V3+V4	192	-	Feature vector #3 combined with Feature vector #4
11	V1+V2+V3	201	-	Feature vector #1 combined with Feature vector #2 and Feature vector #3
12	V1+V2+V4	201	-	Feature vector #1 combined with Feature vector #2 and Feature vector #4
13	V1+V3+V4	237	-	Feature vector #1 combined with Feature vector #3 and Feature vector #4
14	V2+V3+V4	252	-	Feature vector #2 combined with Feature vector #3 and Feature vector #4
15	V1+V2+V3+V4	297	-	Feature vector #1 combined with Feature vector #2, Feature vector #3, and Feature vector #4

- Three dimensions (i.e., x horizontal axis, y vertical axis, and z depth value).

For both experimental designs, we constructed fifteen features vectors (as shown in Table 4.3) from the features discussed in the previous three subsections.

4.6 Classification Methods

We used four learning algorithms to evaluate the proposed features thoroughly. The learning algorithms are (refer to Section 2.2.2): Support Vector Machine (SVM), k-Nearest Neighbor (kNN) with k=3 and k=5, and Random Forest (RF). Tuning the parameters of the previous learning algorithms recommended before proceeding with classification. For example, the performance of SVM classifier affected by the parameter selection. We used SVM with linear function where we have only the complexity parameter C to optimize. C controls the strictness of the support vector margins, a low C value leads to soft margins with

minimum misclassification cost and vice versa. Therefore, the training dataset (V1+V2+V3+V4) used to optimize the following parameters:

- SVM penalty term (C).
- k parameter in kNN.
- Number of trees parameter in RF.

The training dataset was split into 70% for training and 30% for validation. The optimal parameters, obtained by the validation process used for test on unseen data.

The SVM kernel we set to a linear function for two reasons: First, the linear function was applied in similar studies such as [21] and [22]. Second, other kernel functions neglected due to their bad performance.

CHAPTER 5

EXPERIMENTS AND RESULTS

As discussed in the previous chapter, we have two experimental designs, and set of sub experiments in each design. This chapter explained in details the experimental results as well as our analysis of these results.

5.1 Experimental Design 1

In this experimental design, we used the data discussed in Section 4.3.1. Additionally, we preprocessed the data using translation and normalization (refer to Section 4.4). We carried out two experiments:

- **Experiment 1 :** raw joints positions used as features, and the recognition was performed using thresholds based on Dynamic Time Warping (DTW) distance.

- **Experiment 2 :** the features discussed in Section 4.5 were extracted from the data, and the classifiers in Section 4.6 used for recognition.

5.1.1 Experiment 1

In this experiment, we used the raw joints positions to classify the human actions. The baseline study (i.e., EasyGR [2]) used two techniques: Dynamic Time Warping (DTW) and Hidden Markov Model (HMM) to extract thresholds from set of actions. Since DTW outperformed HMM in the baseline study, we used it here for benchmarking. DTW used to align the human joints trajectories. The DTW calculated among all training samples in a certain class then the upper threshold is set to the distance between the most different samples, and the lower threshold is set to zero. The reference action selected from the training samples such that it has the minimum distance to all other training samples. We performed 10 folds thresholds classification using two datasets:

1. Dataset 1:
    - **For each action:** 80 samples recorded by EasyGR team [2].
    - **Total:** 560 samples for all 7 actions.
2. Dataset 2:
    - **For each action:** 60 samples recorded by EasyGR team [2], and 20 samples recorded by our research team.
    - **Total:** 560 samples for all 7 actions.

Table 5.1: DTW Threshold Classification Results

	Dataset 1	**Dataset 2**
Fold 1	92.8571	53.5714
Fold 2	92.8571	57.1429
Fold 3	92.8571	75
Fold 4	94.6429	66.0714
Fold 5	94.6429	69.6429
Fold 6	94.6429	58.9286
Fold 7	94.6429	60.7143
Fold 8	100	67.8571
Fold 9	91.0714	60.7143
Fold 10	96.4286	76.7857
Average	**94.4643**	**64.6429**

The results of DTW threshold classification using the above two test datasets are shown in Table 5.1.

5.1.2 Experiment 2

In this experiment, we carried out three sub experiments, each one related to a particular Daubechies wavelet (db1, db4, and db7). Then, for each sub experiment we extracted the features described in Section 4.5 and classify the data using RF, 3NN, 5NN and SVM classifiers.

For RF, the number of trees was set to 100 which is the default option provided by Weka [87]. For k-NN, The training dataset (V1+V2+V3+V4) split into train data (70%) and validation data (30%) to optimize the k parameter. For SVM, we a linear kernel provided by [88]. To optimize the penalty term (C), the training dataset (V1+V2+V3+V4) was split to train data (70%) and validation data (30%).

The model was trained on EasyGR training dataset (80 samples for each ac-

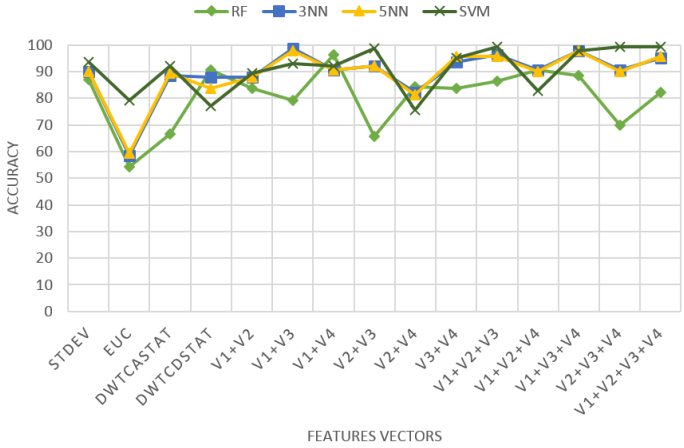
tion, 560 in total), and tested on the dataset recorded by us (20 samples for each action, 140 in total) in order to improve the recognition rate scored in the first experiment shown in Table 5.1. The classifier accuracies for our test dataset (the unseen dataset) are shown in Figure 5.1 and Table 5.2, whereas the F-Measure values are illustrated in Table 5.3.

5.1.3 Discussion

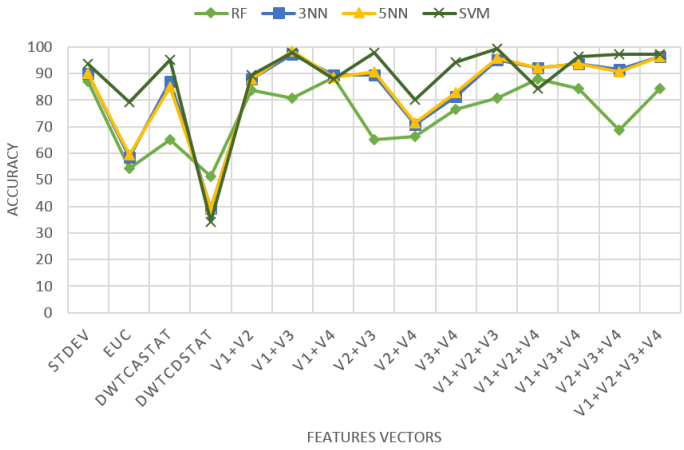
In the first experiment (see Table 5.1), the classification based on raw joints coordinates performed in excellent manner (94.46) with test data that is similar to the training data. However, the performance dropped significantly (64.64) when testing on novel data. The main reason for this performance drop is the variation in the performed subjects, which are not included in dataset 1.

For the second experiment, by looking at the learning curves at Figure 5.1 we can notice a similar start of the curves because the features vectors V1 (standard deviation) and V2 (Euclidean distance) are independent from the change of the mother wavelet. Nevertheless, the features vector V2 represents a performance drop by all classifiers that indicates a low discriminative power of this particular features vector.

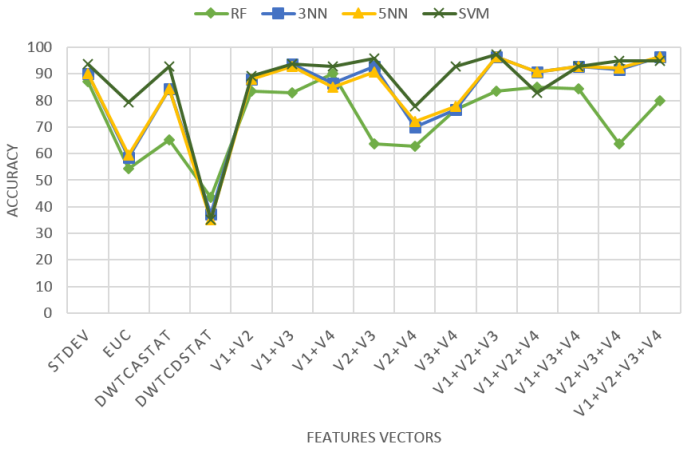
Moreover, another performance drop spotted at features vector V4 (statistical properties of DWT detail coefficients). This performance drop became much sharper as we increase the degree of the mother wavelet (see Figure 5.1). This is justified by the noise generated by DWT detail coefficients as the degree of the



(a) db1



(b) db4



(c) db7

Figure 5.1: Learning Curves for Different Mother Wavelets

mother wavelet increases. This increase of noise negatively affect the discriminative power of this particular features vector.

On the other hand, features vector V3 (statistical properties of DWT approximation coefficients) proves to be a stable performer across all the selected mother wavelets. Since the approximation coefficients inherits a similar shape as the original trajectory, it provides discriminative power to the classifiers, which led to a stable performance.

Generally, combining the features vectors introduces more stability to the learning curves, which illustrates that the discriminative power enhanced. The highest accuracy rates achieved by features combinations that has V3 as a member. This observation demonstrates that DWT approximation coefficients are good descriptors for the human joint trajectory.

From the mother wavelet point of view, by looking at the learning curves at Figure 5.1 we can say that we haven't gain much by increasing the degree of the mother wavelet. Therefore, db1 wavelet is the most suitable mother wavelet for our data (i.e., human joint trajectories).

As for the classifiers, the linear SVM classifier was the best performer across all mother wavelets; also, it scores the highest recognition rate in the design (i.e 99.28). Moreover, both k-NN schemes performed almost identically across all mother wavelets designs, which indicates a low influence of the k parameter. The random forest classifier has a similar pattern as the other classifiers (i.e., same performance drops), but its scores were lower than other classifiers.

Finally, improving the recognition rates from 64.64% in experiment 1 to above 90% in experiment 2, demonstrates that extracted features can outperform the raw 3D joints positions. Therefore, we accept the first hypothesis (H1) and we reject its null hypothesis.

Table 5.2: Classification Accuracy for Features Combinations

	db1				db4				db7			
	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	87.1429	90	90	93.5714	87.1429	90	90	93.5714	87.1429	90	90	93.5714
Euc	54.2857	58.5714	59.2857	79.2857	54.2857	58.5714	59.2857	79.2857	54.2857	58.5714	59.2857	79.2857
DWTcAStat	66.4286	88.5714	89.2857	92.1429	65	87.1429	85	95	65	84.2857	84.2857	92.8571
DWTcDStat	90.7143	87.8571	83.5714	77.1429	51.4286	39.2857	39.2857	34.2857	43.5714	37.1429	35	35
V1+V2	83.5714	87.8571	87.8571	89.2857	83.5714	87.8571	87.8571	89.2857	83.5714	87.8571	87.8571	89.2857
V1+V3	79.2857	98.5714	97.8571	92.8571	80.7143	97.1429	98.5714	97.8571	82.8571	93.5714	92.8571	93.5714
V1+V4	96.4286	90.7143	90.7143	92.1429	88.5714	89.2857	88.5714	87.8571	90	86.4286	85	92.8571
V2+V3	65.7143	92.1429	92.1429	98.5714	65	89.2857	90.7143	97.8571	63.5714	92.8571	90.7143	95.7143
V2+V4	84.2857	82.1429	81.4286	75.7143	66.4286	70.7143	71.4286	80	62.8571	70	72.1429	77.8571
V3+V4	83.5714	93.5714	95.7143	95	76.4286	81.4286	82.8571	94.2857	76.4286	76.4286	77.8571	92.8571
V1+V2+V3	86.4286	96.4286	95.7143	99.2857	80.7143	95	95.7143	99.2857	83.5714	96.4286	96.4286	97.1429
V1+V2+V4	90.7143	90.7143	90	82.8571	87.8571	92.1429	92.1429	84.2857	85	90.7143	90.7143	82.8571
V1+V3+V4	88.57143	97.85714	97.85714	97.8571	84.28571	93.57143	93.57143	96.4286	84.28571	92.85714	92.85714	92.8571
V2+V3+V4	70	90.7143	90	99.2857	68.5714	91.4286	90.7143	97.1429	63.5714	91.4286	92.1429	95
V1+V2+V3+V4	82.1429	95	95.7143	99.2857	84.2857	96.4286	96.4286	97.1429	80	96.4286	96.4286	95

Table 5.3: F-Measure Values for Extracted Features in Experimental Design 1

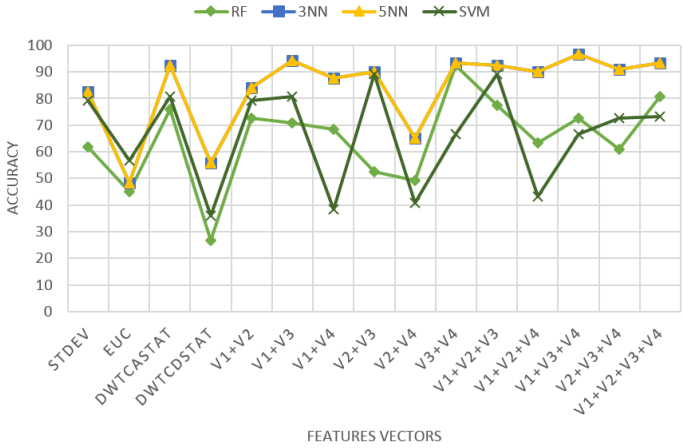
		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	Circle	0.8	0.9756	0.9744	0.9756	0.8	0.9756	0.9744	0.9756	0.8	0.9756	0.9744	0.9756
	Elongation	0.9189	0.5714	0.5714	0.75	0.9189	0.5714	0.5714	0.75	0.9189	0.5714	0.5714	0.75
	Swim	0.6667	0.95	0.9756	0.9744	0.6667	0.95	0.9756	0.9744	0.6667	0.95	0.9756	0.9744
	Punch	0.8571	1	1	1	0.8571	1	1	1	0.8571	1	1	1
	Smash	1	0.9744	0.9744	1	1	0.9744	0.9744	1	1	0.9744	0.9744	1
	SwipeLeft	1	0.7692	0.7547	0.8333	1	0.7692	0.7547	0.8333	1	0.7692	0.7547	0.8333
	SwipeRight	0.8333	1	1	1	0.8333	1	1	1	0.8333	1	1	1
Euc	Circle	0.8889	1	1	1	0.8889	1	1	1	0.8889	1	1	1
	Elongation	0	0.0952	0.0952	0.9189	0	0.0952	0.0952	0.9189	0	0.0952	0.0952	0.9189
	Swim	0.5	0.5116	0.5641	0.5455	0.5	0.5116	0.5641	0.5455	0.5	0.5116	0.5641	0.5455
	Punch	0.2143	0.1481	0.1875	0.7619	0.2143	0.1481	0.1875	0.7619	0.2143	0.1481	0.1875	0.7619
	Smash	0.4615	0.5714	0.5714	0.4444	0.4615	0.5714	0.5714	0.4444	0.4615	0.5714	0.5714	0.4444
	SwipeLeft	0.6557	0.6452	0.6452	0.8333	0.6557	0.6452	0.6452	0.8333	0.6557	0.6452	0.6452	0.8333
	SwipeRight	0.6349	0.678	0.6897	0.9524	0.6349	0.678	0.6897	0.9524	0.6349	0.678	0.6897	0.9524
DWTcAStat	Circle	0	0.8947	0.8837	0.8163	0.0952	0.8889	0.8571	0.9524	0.1818	0.5333	0.5333	0.9756
	Elongation	0.8163	1	1	0.9756	0.7018	1	1	0.9302	0.625	1	1	0.8889
	Swim	0.0952	0.5185	0.4615	0.7097	0.2609	0.4615	0.3333	0.8889	0.1818	0.6667	0.6667	0.9744
	Punch	0.8889	1	1	0.9756	0.9756	1	1	0.9524	0.9744	1	1	0.9091
	Smash	0.8889	1	1	0.9474	0.6667	1	1	0.9189	0.6667	1	1	0.7097
	SwipeLeft	0.5405	0.7273	0.7843	1	0.5634	0.6897	0.6557	1	0.625	0.6667	0.6667	1
	SwipeRight	0.9091	1	1	1	0.9189	1	1	1	0.9231	1	1	1
DWTcDStat	Circle	1	1	1	0.9744	0.5946	0.3448	0.0833	0.4	0.129	0.1538	0	0.4615
	Elongation	0.9744	0.8571	0.8235	0.6667	0.4416	0.5862	0.6415	0.5714	0.5902	0.6032	0.5846	0.5769
	Swim	0.766	0.8163	0.7347	0.7308	0.5625	0.0476	0.1429	0.2449	0.4211	0.1395	0.0571	0.1702
	Punch	0.9524	0.7619	0.7442	0.7442	0.7826	0.4286	0.45	0.3125	0.6667	0.3529	0.3636	0.4
	Smash	0.7097	0.7429	0.7059	0.7805	0.2222	0.5	0.4783	0.2791	0.0909	0.8108	0.75	0.2917
	SwipeLeft	0.9756	1	0.9302	0.8718	0.6667	0.4889	0.52	0.4444	0.4091	0.2	0.1905	0.3784
	SwipeRight	0.95	0.9744	0.9189	0.6111	0.0909	0.1667	0.08	0.069	0.25	0	0	0
V1+V2	Circle	0.8333	1	1	1	0.8333	1	1	1	0.8333	1	1	1
	Elongation	0.7097	0.3333	0.2609	0.9474	0.7097	0.3333	0.2609	0.9474	0.7097	0.3333	0.2609	0.9474
	Swim	0.7097	1	1	0.6486	0.7097	1	1	0.6486	0.7097	1	1	0.6486
	Punch	0.8571	0.9744	1	1	0.8571	0.9744	1	1	0.8571	0.9744	1	1
	Smash	0.9756	1	1	0.8333	0.9756	1	1	0.8333	0.9756	1	1	0.8333
	SwipeLeft	0.8	0.7143	0.7018	0.8333	0.8	0.7143	0.7018	0.8333	0.8	0.7143	0.7018	0.8333
	SwipeRight	0.9091	0.9756	1	0.9756	0.9091	0.9756	1	0.9756	0.9091	0.9756	1	0.9756
V1+V3	Circle	0.8889	0.9744	0.9474	0.8333	0.7826	0.9474	0.9744	0.9524	0.9474	0.75	0.7097	0.9756
	Elongation	0.6531	0.9744	0.9744	0.9756	0.6471	0.9474	0.9744	1	0.6531	0.9744	0.9744	0.9091
	Swim	0.5714	1	1	0.75	0.6207	1	1	0.9474	0.5714	1	1	0.9744
	Punch	0.8235	1	1	0.9756	0.8571	1	1	0.9756	0.8571	1	1	0.9091
	Smash	0.7879	1	1	0.9474	1	1	1	0.9744	0.9744	1	1	0.75
	SwipeLeft	0.8889	0.9524	0.9302	1	0.7843	0.9091	0.9524	1	0.8696	0.8163	0.8	1
	SwipeRight	0.8696	1	1	1	0.8889	1	1	1	0.8889	1	1	1
V1+V4	Circle	0.9756	1	1	0.9744	0.9744	0.8571	0.8235	0.8571	1	0.7879	0.7097	1
	Elongation	0.8889	0.6207	0.6207	0.75	0.7826	0.6667	0.6667	0.6667	0.6857	0.6667	0.6667	0.6667
	Swim	0.9744	0.9524	0.9524	1	0.6667	1	1	0.9524	0.8235	0.9756	0.9756	1
	Punch	1	0.9524	0.9302	0.95	0.8947	0.9756	0.9756	0.9524	1	0.95	0.9268	1
	Smash	0.9756	0.9474	0.9474	1	0.9756	1	1	1	0.9302	0.95	0.95	1
	SwipeLeft	0.9524	0.8333	0.8511	0.8511	0.9524	0.7407	0.7273	0.7547	0.9756	0.7143	0.7018	0.8511
	SwipeRight	0.9756	0.9756	0.9756	0.9048	0.9091	1	1	0.9474	0.8511	1	1	0.9302
V2+V3	Circle	0.8182	1	1	0.9756	0.6667	0.9524	0.9302	1	0.8571	1	1	1
	Elongation	0.7879	1	0.9744	1	1	0.9189	1	1	0.6667	1	0.9474	1
	Swim	0.1818	0.6667	0.6667	0.9744	0.0952	0.5714	0.5185	1	0.0952	0.7097	0.6667	1
	Punch	0.3636	0.9756	1	0.9756	0.375	1	1	0.9302	0.4138	0.9756	0.9756	0.8696
	Smash	0.7647	0.9744	1	0.9744	0.8235	1	1	0.9189	0.9189	0.9744	0.9744	0.8235
	SwipeLeft	0.6667	0.8	0.7843	1	0.5797	0.7547	0.8	1	0.5405	0.8163	0.7692	1
	SwipeRight	0.7407	1	1	1	0.7407	1	1	1	0.7407	1	1	1
V2+V4	Circle	1	1	1	1	0.7879	0.9189	0.9744	1	0.5625	0.9189	0.9474	1
	Elongation	0.8889	0.4615	0.4615	0.7879	0.6875	0.5714	0.5714	0.9474	0.5806	0.6667	0.6667	0.8571
	Swim	0.7692	0.7568	0.7429	0.64	0.4211	0.5263	0.5714	0.5946	0.3846	0.4545	0.4762	0.5417
	Punch	0.7097	0.8889	0.8889	0.68	0.3333	0.4	0.4828	0.7451	0.4	0.3333	0.4615	0.7273
	Smash	0.8889	0.9744	0.9744	0.1818	0.7879	0.9744	0.9189	0.32	0.8889	0.9474	0.9474	0.4828
	SwipeLeft	0.9302	0.7018	0.6897	0.8696	0.9091	0.7018	0.6897	0.8163	0.8163	0.7692	0.7273	0.8889
	SwipeRight	0.7273	0.8889	0.8696	0.9231	0.7018	0.7143	0.7037	1	0.6909	0.6909	0.7451	0.9231
V3+V4	Circle	0.9744	1	1	0.8889	0.5185	0.5714	0.6207	0.9302	0.6207	0.3333	0.4	1
	Elongation	0.7273	1	1	0.9756	0.7692	1	1	0.9756	0.7407	1	1	0.9302
	Swim	0.1818	0.7879	0.8571	0.8571	0.4615	0.5185	0.5185	0.9189	0.3333	0.4615	0.4615	1
	Punch	1	0.9474	0.9744	0.9744	0.8235	1	1	0.9091	1	0.9189	0.9474	0.8511
	Smash	0.8889	0.9524	0.9756	0.95	0.9302	0.9756	1	0.8571	0.8235	0.9302	0.9524	0.6667
	SwipeLeft	0.8333	0.8511	0.8889	1	0.7407	0.6452	0.6557	1	0.678	0.5882	0.597	1
	SwipeRight	1	1	1	1	0.9091	0.9048	0.9302	1	1	0.9524	0.9524	1
V1+V2+V3	Circle	0.8261	1	1	1	0.75	1	1	1	0.8	1	1	1
	Elongation	0.9	0.8571	0.8235	1	0.6667	0.8235	0.8235	1	0.8889	0.8571	0.8571	1
	Swim	0.6667	1	1	1	0.6667	0.9744	1	1	0.5185	1	1	1
	Punch	0.8235	1	1	0.9756	0.8571	1	1	0.9756	0.8235	1	1	0.9091
	Smash	0.9756	1	1	0.9744	1	1	1	0.9744	0.9524	1	1	0.8889
	SwipeLeft	0.9302	0.8889	0.8696	1	0.7692	0.8511	0.8696	1	0.8889	0.8889	0.8889	1
	SwipeRight	0.8696	1	1	1	0.8889	1	1	1	0.8696	1	1	1
V1+V2+V4	Circle	0.9524	1	1	1	0.9524	1	1	1	0.8444	1	1	1
	Elongation	0.7647	0.5714	0.5185	0.75	0.7647	0.6207	0.6207	0.9189	0.6452	0.6207	0.6207	0.8889
	Swim	1	0.9756	0.9756	0.7083	0.8947	1	1	0.6471	0.9189	0.9744	0.9744	0.619
	Punch	0.8571	1	1	0.8182	0.8	1	1	0.8511	0.8	0.9744	0.9744	0.8889
	Smash	0.9744	0.9744	0.9744	0.6667	0.9744	1	1	0.6452	0.9744	1	1	0.5806
	SwipeLeft	0.9756	0.7692	0.7547	0.8511	0.9524	0.8	0.7843	0.7843	0.9091	0.7692	0.7692	0.8333
	SwipeRight	0.8163	1	1	0.9744	0.8	0.9756	1	1	0.8163	0.9756	0.9756	0.9474
V1+V3+V4	Circle	0.8511	1	1	0.9524	0.9524	0.75	0.7879	0.9756	0.9474	0.7097	0.7097	1
	Elongation	0.8889	0.9189	0.9189	0.9756	0.5128	0.9744	0.9474	0.9756	0.7692	0.9744	0.9744	0.9302
	Swim	0.7097	1	1	0.9474	0.6667	1	1	0.9744	0.5185	1	1	1
	Punch	0.9189	1	1	1	0.9474	1	1	0.9302	0.8235	1	1	0.8511
	Smash	1	1	1	0.9744	1	1	1	0.8889	0.9744	1	1	0.6667
	SwipeLeft	0.8696	0.9302	0.9302	1	0.8163	0.8163	0.8163	1	0.9091	0.8	0.8	1
	SwipeRight	0.9302	1	1	1	0.9524	1	1	1	0.8696	1	1	1
V2+V3+V4	Circle	0.9744	1	1	1	0.6957	1	1	1	0.8649	0.9756	0.9756	1
	Elongation	0.8293	0.8571	0.8571	1	0.75	0.9474	0.9189	1	0.5714	1	1	1
	Swim	0	0.75	0.7097	1	0	0.6667	0.6667	1	0	0.6207	0.6207	1
	Punch	0.4615	1	1	0.9756	0.6	1	1	0.9091	0.6471	0.9744	1	0.8511
	Smash	0.8889	1	1	0.9744	0.95	1	1	0.8889	0.75	1	1	0.7879
	SwipeLeft	0.625	0.7547	0.7407	1	0.6667	0.7692	0.7547	1	0.5479	0.8	0.8	1
	SwipeRight	0.7407	1	1	1	0.7692	1	1	1	0.8163	0.9756	1	1
V1+V2+V3+V4	Circle	0.8	1	1	1	0.8163	1	1	1	0.8261	1	1	1
	Elongation	0.8571	0.8235	0.8235	1	0.8235	0.8571	0.8571	1	0.65	0.8571	0.8571	1
	Swim	0.4	0.9744	1	1	0.5714	1	1	1	0.4615	1	1	1
	Punch	0.8571	1	1	0.9756	0.8889	1	1	0.9091	0.8571	1	1	0.8511
	Smash	1	1	1	0.9744	0.9524	1	1	0.8889	0.95	1	1	0.7879
	SwipeLeft	0.8	0.8511	0.8696	1	0.8511	0.8889	0.8889	1	0.8333	0.8889	0.8889	1
	SwipeRight	0.8889	1	1	1	0.9091	1	1	1	0.8889	1	1	1

5.2 Experimental Design 2

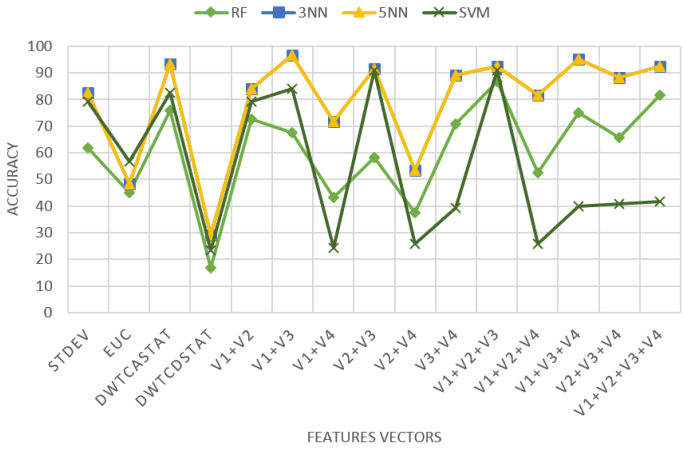
Three experiments were carried out using data described in Section 4.3.2, where the synthetic data dedicated only for training, and real data dedicated only for testing. Furthermore, the preprocessing techniques discussed in Section 4.4 were applied to aforementioned data. The experiments designed to inspect three different quantization schemes:

- **Experiment 1 :** conducted without performing quantization preprocessing.
- **Experiment 2 :** a 3D cube with dimensions ($x \in [-3, 3]$, $y \in [-3, 3]$, and $z \in [-3, 3]$) was placed around the subject (refer to Figure 4.9). The distance between centers set to one unit on Kinect coordinate system (i.e., one meter). Thus, the number of centers was 343 (i.e., $7 \times 7 \times 7$).
- **Experiment 3 :** a 3D cube with dimensions ($x \in [-3, 3]$, $y \in [-3, 3]$, and $z \in [-3, 3]$) was placed around the subject (refer to Figure 4.9). The distance between centers set to half unit on Kinect coordinate system (i.e., half meter). Thus, the number of centers was 2197 (i.e., $13 \times 13 \times 13$).

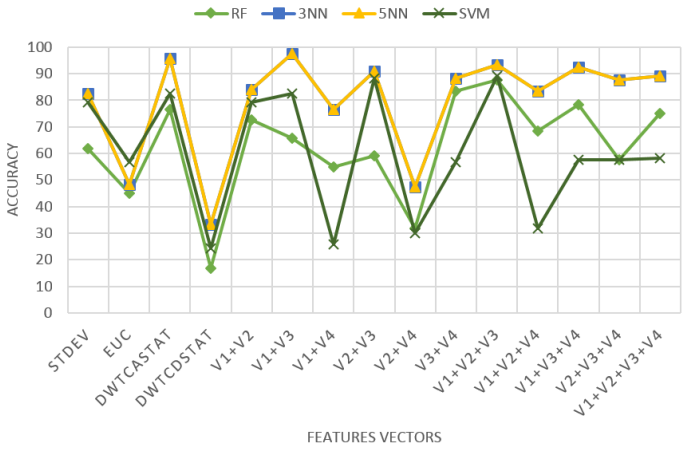
In all of experiments above, we extracted the features listed in Section 4.5 using three mother wavelets: db1, db4, and db7, the classifiers used are RF, 3NN, 5NN, and SVM. Figure 5.2, Figure 5.3, and Figure 5.4 show the classifiers accuracy of experiments 1, 2, and 3 respectively. Moreover, at the end of this chapter you can find Tables 5.4, 5.6, and 5.8 that show the numerical values of the classifiers' accuracy, whereas Tables 5.5, 5.7, and 5.9 show values of classification F-Measure.



(a) db1

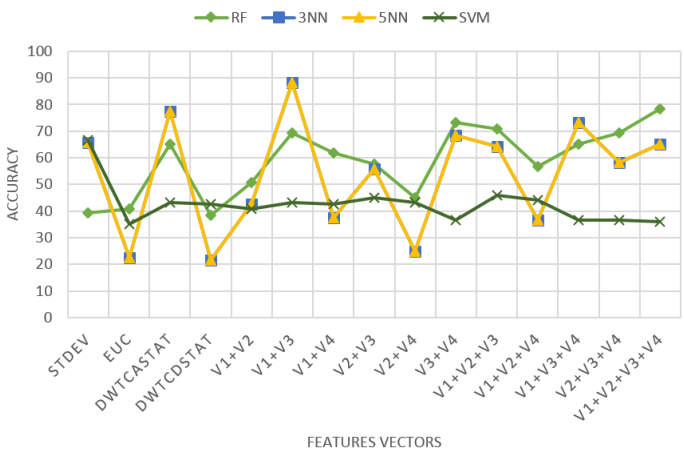


(b) db4

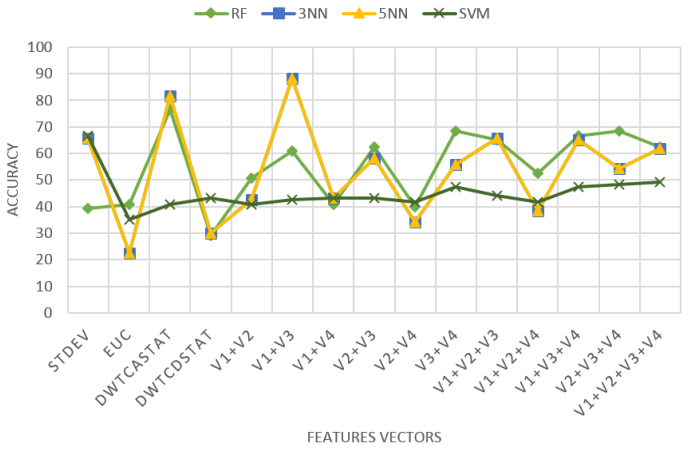


(c) db7

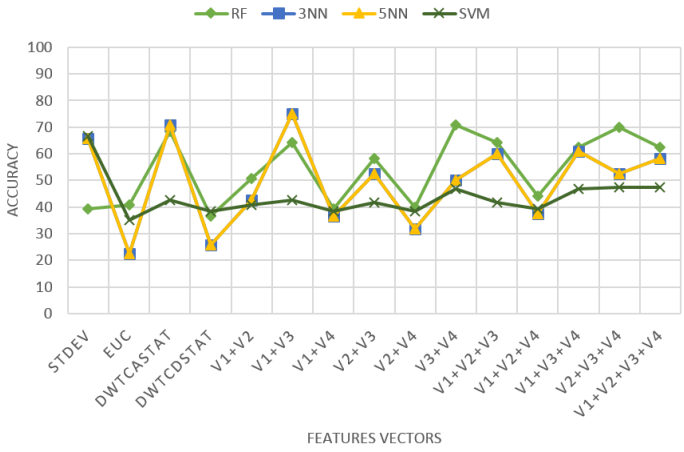
Figure 5.2: Learning Curves for Different Mother Wavelets with no Quantization



(a) db1

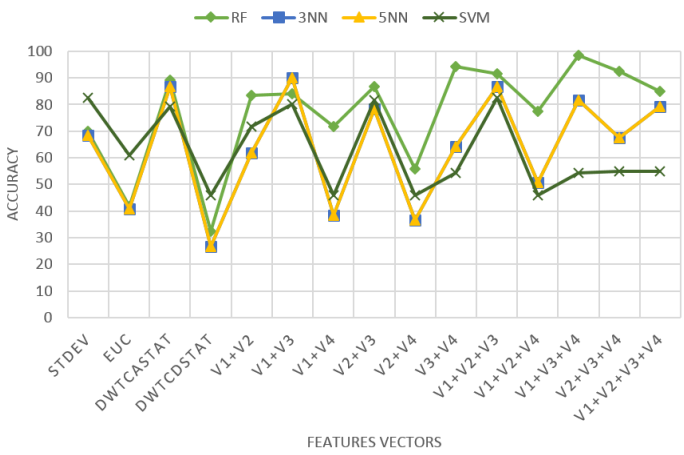


(b) db4

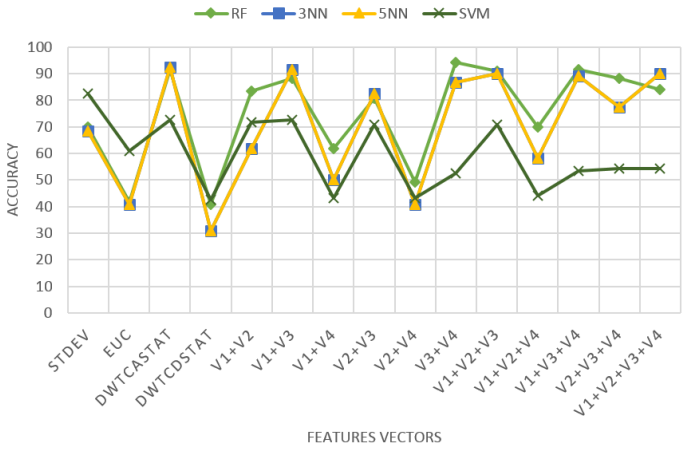


(c) db7

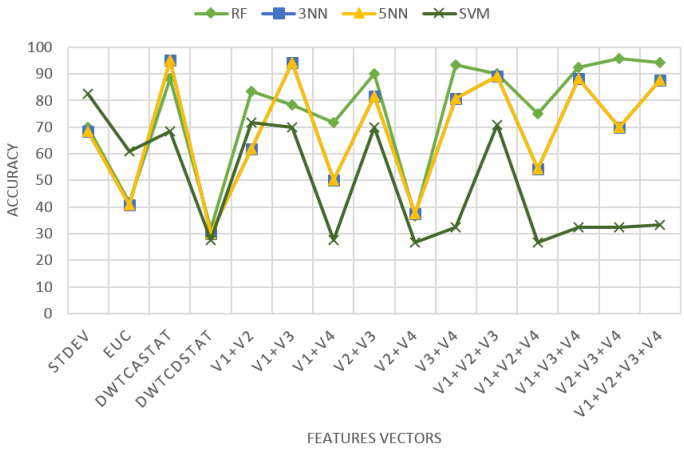
Figure 5.3: Learning Curves for Different Mother Wavelets with Quantization Unit = 1



(a) db1



(b) db4



(c) db7

Figure 5.4: Learning Curves for Different Mother Wavelets with Quantization Unit = 0.5

5.2.1 Discussion

In the first experiment using db1 as mother wavelet (see Figure 5.2(a)), the classifiers exhibited a moderate performance for V1 (standard deviation), then their performance dropped for V2 (Euclidean distance). This observation does not change for the next sub experiments (db4 and db7), since V1 and V2 are independent from the selection of the mother wavelet.

Afterwards, the performance of the classifiers increased using V3 (statistical properties DWT approximation), since this features vector preserved the same pattern of the original trajectory (please refer to Figure 4.19). As for V4 (statistical properties of DWT detail), the classifiers performance dropped significantly which indicate that this features vector is not a good descriptor by itself. For the rest of the combinations, we observed enhancements in classifiers performance with some peaks at V1+V3, V3+V4, V1+V3+V4, and V1+V2+V3+V4, also some drops were observed at V1+V4, V2+V4, and V1+V2+V4.

Moreover, in the other sub experiments (see Figure 5.2(b) and Figure 5.2(c)), we experienced a similar pattern to the one appeared in Figure 5.2(a), except the drops at: V1+V4, V2+V4, and V1+V2+V4 became sharper. These sharp drops were caused by the noise of DWT detail coefficients as we increase the degree of the mother wavelet (please refer to Figure 4.19). Nonetheless, this was not the case with DWT approximation coefficients, since all classifiers scored higher recognition rates with DWT approximation coefficients as we increase the degree of the mother wavelet. As a result, we get a maximum score 97.5 of accuracy

in experiment1 using k-NN classifiers with features vector (V1+V3) and db7 as mother wavelet.

Moving to the second experiment (quantization with full unit), using db1 as mother wavelet (see Figure 5.3(a)), we noticed significant drops at: V2, V4, V1+V4, V2+V4, and V1+V2+V4 which seems similar behavior seen in the first experiment. Apparently, the new quantization scheme (quantization unit = one unit) does not make them any better. kNN held its place as top performer with scores of 77.5 and 88.33 for features vectors V3 and V1+V3 respectively.

In the sub experiment using db4 as mother wavelet (see Figure 5.3(b)), the kNN performance does not change much from the previous sub experiment using db1 as mother wavelet. Whereas, RF classifier exhibited less sensitivity to the change in the features vectors, with a drop in its mean recognition rate of 58.72 in the previous sub experiment to be 54.94 in this sub experiment.

In the final sub experiment using db7 as mother wavelet (see Figure 5.3(c)), all classifiers became less sensitive to the change in the features vectors. The performance degradation became much severe as we increase the degree of the mother wavelet, mainly due to the noise of DWT detail coefficients as we increase the degree of the mother wavelet (please refer to Figure 4.19).

In general, the outcomes of the second experiment (quantization with full unit) were not up to the expectations. The undesirable performance of the second experiment can be justified by the significant change of the joints positions due to a large quantization unit, which led to major information loss. This observation

clearly demonstrated by the steady performance of SVM classifier throughout this experiment. The SVM maintained a consistent performance (compared to the first experiment) ignoring the change of the features vectors, which indicates that the classifier does not get enough information to classify the instances.

As for the final experiment (quantization with half unit), using db1 as mother wavelet (see Figure 5.4(a)), we can notice that RF moved to be the top performer meaning that it responds well to the quantization preprocessing. The classifiers performance dropped at features vectors V2, V4, V1+V4, V2+V4, and V1+V2+V4. Nevertheless, combining DWT approximation and detail coefficients can enhance the performance with db1 and half unit quantization, as we can see from the performance peaks at V3+V4 and V1+V3+V4.

When we implemented db4 and db7 as mother wavelets (see Figure 5.4(b) and Figure 5.4(c)), the performance of RF classifier dropped and moved close to the performance of kNN classifiers. The performance patterns observed at db1 continued to persist for db4 and db7, with the performance drop at V4 became sharper for the wavelets with higher degrees.

This experiment (quantization with half unit), overcomes the weaknesses of the first experiment by moving the joints positions from a continuous space into a predefined quantization groups. In addition, it seems to be a reasonable combination of distance between centers and centers resolution. Therefore, it increases the quantization groups and provides a better description of the joints positions.

The maximum recognition rates achieved by the classifiers:

- RF classifier was 98.33 using features vector (V1+V3+V4) with db1 wavelet and half unit of quantization.

- 3NN and 5NN classifiers has their maximum score (97.5) using features vector (V1+V3) with db7 wavelet and no quantization. Remarkably, 3NN and 5NN classifiers performed in a similar manner, which indicate a low significance of k parameter (further investigation is needed).

- Linear SVM has a highest score of 90.83 using features vector (V1+V2+V3) with db4 wavelet and no quantization.

As a result, we get our maximum recognition rate using quantization preprocessing, Noteworthy among the maximum recognition rates mentioned earlier, they all share the involvement of V1 (standard deviation) and V3 (DWT approximation). This demonstrates a high discrimination power for these two features vectors.

The recognition rates shown in Figure 5.2, Figure 5.3, and Figure 5.4 (with maximum = 98.33) demonstrates that a real human action can be recognized using a classifier that was trained on synthetic human action data. Therefore, we accept the second hypothesis (H2) and we reject its null hypothesis.

Table 5.4: Classification Accuracy for Features Vectors with no Quantization

	db1				db4				db7			
	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	61.6667	82.5	82.5	79.1667	61.6667	82.5	82.5	79.1667	61.6667	82.5	82.5	79.1667
Euc	45	48.3333	48.3333	56.6667	45	48.3333	48.3333	56.6667	45	48.3333	48.3333	56.6667
DWTcAStat	75.8333	92.5	92.5	80.8333	75.8333	93.3333	93.3333	82.5	76.6667	95.8333	95.8333	82.5
DWTcDStat	26.6667	55.8333	55.8333	35.8333	16.6667	29.1667	29.1667	23.3333	16.6667	33.3333	33.3333	24.1667
V1+V2	72.5	84.1667	84.1667	79.1667	72.5	84.1667	84.1667	79.1667	72.5	84.1667	84.1667	79.1667
V1+V3	70.8333	94.1667	94.1667	80.8333	67.5	96.6667	96.6667	84.1667	65.8333	97.5	97.5	82.5
V1+V4	68.3333	87.5	87.5	38.3333	43.3333	71.6667	71.6667	24.1667	55	76.6667	76.6667	25.8333
V2+V3	52.5	90	90	89.1667	58.3333	91.6667	91.6667	90.8333	59.1667	90.8333	90.8333	88.3333
V2+V4	49.1667	65	65	40.8333	37.5	53.3333	53.3333	25.8333	31.6667	47.5	47.5	30
V3+V4	92.5	93.3333	93.3333	66.6667	70.8333	89.1667	89.1667	39.1667	83.3333	88.3333	88.3333	56.6667
V1+V2+V3	77.5	92.5	92.5	89.1667	86.6667	92.5	92.5	90.8333	87.5	93.3333	93.3333	89.1667
V1+V2+V4	63.3333	90	90	43.3333	52.5	81.6667	81.6667	25.8333	68.3333	83.3333	83.3333	31.6667
V1+V3+V4	72.5	96.66667	96.66667	66.6667	75	95	95	40	78.33333	92.5	92.5	57.5
V2+V3+V4	60.8333	90.8333	90.8333	72.5	65.8333	88.3333	88.3333	40.8333	57.5	87.5	87.5	57.5
V1+V2+V3+V4	80.8333	93.3333	93.3333	73.3333	81.6667	92.5	92.5	41.6667	75	89.1667	89.1667	58.3333

Table 5.5: Classification F-Measure for Features Vectors with no Quantization

		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	Balance	0.5714	0.8889	0.8889	0.8571	0.5714	0.8889	0.8889	0.8571	0.5714	0.8889	0.8889	0.8571
	Jump	0.8889	1	1	0.6286	0.8889	1	1	0.6286	0.8889	1	1	0.6286
	Kick	0.2857	0.5882	0.5882	0.6207	0.2857	0.5882	0.5882	0.6207	0.2857	0.5882	0.5882	0.6207
	Pickup	0.8163	0.9756	0.9756	0.8163	0.8163	0.9756	0.9756	0.8163	0.8163	0.9756	0.9756	0.8163
	Punch	0.6349	0.7143	0.7143	0.8511	0.6349	0.7143	0.7143	0.8511	0.6349	0.7143	0.7143	0.8511
	Stretch	0.3448	0.7879	0.7879	0.8889	0.3448	0.7879	0.7879	0.8889	0.3448	0.7879	0.7879	0.8889
Euc	Balance	0.1429	0.1739	0.1739	0.6452	0.1429	0.1739	0.1739	0.6452	0.1429	0.1739	0.1739	0.6452
	Jump	0.7273	0.4762	0.4762	0.4211	0.7273	0.4762	0.4762	0.4211	0.7273	0.4762	0.4762	0.4211
	Kick	0.3902	0.4324	0.4324	0.4375	0.3902	0.4324	0.4324	0.4375	0.3902	0.4324	0.4324	0.4375
	Pickup	0.4727	0.6087	0.6087	0.7895	0.4727	0.6087	0.6087	0.7895	0.4727	0.6087	0.6087	0.7895
	Punch	0.2308	0	0	0.2222	0.2308	0	0	0.2222	0.2308	0	0	0.2222
	Stretch	0.5614	0.9048	0.9048	0.8696	0.5614	0.9048	0.9048	0.8696	0.5614	0.9048	0.9048	0.8696
DWTcAStat	Balance	0.6667	1	1	1	0.6207	0.9756	0.9756	0.9091	0.8235	0.9756	0.9756	0.8571
	Jump	0.5161	0.8163	0.8163	0.6349	0.5806	0.8511	0.8511	0.6786	0.2759	0.9091	0.9091	0.6154
	Kick	0.7647	0.8571	0.8571	0.6667	0.8372	0.8571	0.8571	0.75	0.9048	0.9189	0.9189	0.75
	Pickup	0.7755	0.8889	0.8889	0.5185	0.6522	0.9189	0.9189	0.5714	0.625	0.9474	0.9474	0.7647
	Punch	0.9231	1	1	1	1	1	1	1	0.9756	1	1	1
	Stretch	0.8333	1	1	1	0.7843	1	1	1	0.8696	1	1	1
DWTcDStat	Balance	0	0.0952	0.0952	0	0	0.1667	0.1667	0.2941	0	0.0952	0.0952	0.3684
	Jump	0.1429	0.6897	0.6897	0.6207	0	0	0	0.1176	0	0.0741	0.0741	0.2951
	Kick	0.4082	0.8333	0.8333	0.3333	0.2857	0.3883	0.3883	0.1509	0.2742	0.396	0.396	0.0571
	Pickup	0.0952	0.4255	0.4255	0.4103	0	0	0	0.25	0	0	0	0
	Punch	0.4	0.5455	0.5455	0.2941	0	0.5532	0.5532	0.338	0.1667	0.6512	0.6512	0.2162
	Stretch	0.0833	0.4242	0.4242	0.1379	0	0	0	0.1667	0	0.2857	0.2857	0.3556
V1+V2	Balance	0.8571	0.8571	0.8571	0.8235	0.8571	0.8571	0.8571	0.8235	0.8571	0.8571	0.8571	0.8235
	Jump	0.7273	0.9474	0.9474	0.7442	0.7273	0.9474	0.9474	0.7442	0.7273	0.9474	0.9474	0.7442
	Kick	0.5806	0.5625	0.5625	0.5333	0.5806	0.5625	0.5625	0.5333	0.5806	0.5625	0.5625	0.5333
	Pickup	0.7407	0.8889	0.8889	0.8947	0.7407	0.8889	0.8889	0.8947	0.7407	0.8889	0.8889	0.8947
	Punch	0.7273	0.76	0.76	0.8163	0.7273	0.76	0.76	0.8163	0.7273	0.76	0.76	0.8163
	Stretch	0.6875	1	1	0.8696	0.6875	1	1	0.8696	0.6875	1	1	0.8696
V1+V3	Balance	0.7879	1	1	1	0.8235	1	1	0.9524	0.6207	1	1	0.8571
	Jump	0.3871	0.8511	0.8511	0.6349	0.3333	0.9091	0.9091	0.7018	0.0952	0.9302	0.9302	0.6154
	Kick	0.8108	0.8571	0.8571	0.6207	0.7692	0.8889	0.8889	0.75	0.6957	0.9189	0.9189	0.75
	Pickup	0.6349	0.9474	0.9474	0.5714	0.678	1	1	0.6207	0.5797	1	1	0.7647
	Punch	0.8889	1	1	1	0.7407	1	1	1	0.9524	1	1	1
	Stretch	0.7097	1	1	1	0.5333	1	1	1	0.7879	1	1	1
V1+V4	Balance	0.4615	0.75	0.75	0	0.2609	0.6667	0.6667	0.2941	0.4	0.6667	0.6667	0.3684
	Jump	1	0.9756	0.9756	0.6441	0.6667	0.9048	0.9048	0.1176	0.8718	0.9091	0.9091	0.3279
	Kick	0.5517	0.8085	0.8085	0.3729	0.3125	0.6957	0.6957	0.1852	0.3333	0.6047	0.6047	0.0588
	Pickup	0.8947	0.9048	0.9048	0.4103	0.0952	0.8947	0.8947	0.25	0.7895	0.8889	0.8889	0
	Punch	0.7273	0.9302	0.9302	0.3429	0.6909	0.6667	0.6667	0.3429	0.625	0.7407	0.7407	0.2632
	Stretch	0.2609	0.8571	0.8571	0.1429	0	0.3333	0.3333	0.1667	0	0.7879	0.7879	0.3556
V2+V3	Balance	0.32	0.9189	0.9189	1	0.2963	0.9474	0.9474	1	0.1818	0.9189	0.9189	0.9268
	Jump	0.7778	0.8085	0.8085	0.7547	0.8421	0.8261	0.8261	0.7843	0.75	0.8261	0.8261	0.72
	Kick	0.5	0.75	0.75	0.6207	0.5455	0.7879	0.7879	0.7097	0.7692	0.7879	0.7879	0.75
	Pickup	0.52	0.9756	0.9756	0.9474	0.6047	0.9756	0.9756	0.9474	0.52	0.9756	0.9756	0.9189
	Punch	0.2308	0.9756	0.9756	1	0.4375	0.9756	0.9756	1	0.5556	0.9756	0.9756	1
	Stretch	0.6102	0.9524	0.9524	1	0.6429	0.9756	0.9756	1	0.623	0.9524	0.9524	1

		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
V2+V4	Balance	0	0.1818	0.1818	0	0	0.1818	0.1818	0.303	0	0.1818	0.1818	0.3784
	Jump	0.8	0.6032	0.6032	0.5797	0.6275	0.4286	0.4286	0.1212	0.7429	0.4407	0.4407	0.3158
	Kick	0.4688	0.8718	0.8718	0.36	0.2989	0.5	0.5	0.2642	0.2917	0.4151	0.4151	0.1143
	Pickup	0.55	0.7308	0.7308	0.4737	0.5517	0.7273	0.7273	0.25	0.5946	0.6875	0.6875	0.0833
	Punch	0.16	0.1667	0.1667	0.2286	0.1538	0.3158	0.3158	0.3288	0	0.0606	0.0606	0.2927
	Stretch	0.5652	0.95	0.95	0.5	0.4444	0.9744	0.9744	0.1667	0	0.9268	0.9268	0.4783
V3+V4	Balance	0.8889	0.9189	0.9189	0.3333	0.3333	0.7429	0.7429	0.439	0.7879	0.6667	0.6667	0.4571
	Jump	0.9048	0.8889	0.8889	0.5556	0.6667	0.8	0.8	0.2	0.7317	0.8511	0.8511	0.5313
	Kick	0.9	1	1	0.7778	0.7647	1	1	0.5366	0.8889	1	1	0.4667
	Pickup	0.878	0.8333	0.8333	0.6667	0.6452	0.8235	0.8235	0.2308	0.8235	0.8889	0.8889	0.3077
	Punch	1	0.9756	0.9756	0.8182	0.8163	0.9756	0.9756	0.5373	0.8163	0.8511	0.8511	0.8
	Stretch	0.9756	0.9756	0.9756	0.8235	0.8444	1	1	0.16	0.9474	1	1	0.7111
V1+V2+V3	Balance	0.625	0.9744	0.9744	1	0.9189	0.9744	0.9744	1	0.7097	0.9744	0.9744	0.9268
	Jump	0.8889	0.8261	0.8261	0.7547	0.8235	0.8261	0.8261	0.7843	0.9	0.8444	0.8444	0.7347
	Kick	0.5405	0.7879	0.7879	0.6207	0.8108	0.7879	0.7879	0.7097	0.9	0.8235	0.8235	0.75
	Pickup	0.7407	0.9756	0.9756	0.9474	0.8333	0.9756	0.9756	0.9474	0.8163	0.9756	0.9756	0.9474
	Punch	0.8293	1	1	1	0.8696	1	1	1	0.9744	1	1	1
	Stretch	1	0.9756	0.9756	1	0.9474	0.9756	0.9756	1	0.9268	0.9756	0.9756	1
V1+V2+V4	Balance	0.3846	0.75	0.75	0	0.3333	0.6207	0.6207	0.303	0.6207	0.6667	0.6667	0.3784
	Jump	0.7568	0.9268	0.9268	0.5797	0.7407	0.95	0.95	0.1212	0.9268	0.9744	0.9744	0.3448
	Kick	0.5455	0.8718	0.8718	0.3673	0.3881	0.6316	0.6316	0.2692	0.5246	0.6316	0.6316	0.1176
	Pickup	0.7843	0.9091	0.9091	0.4737	0.5714	0.9756	0.9756	0.24	0.8649	0.9756	0.9756	0.0833
	Punch	0.7	0.9091	0.9091	0.3889	0.6512	0.717	0.717	0.3288	0.7234	0.7308	0.7308	0.3415
	Stretch	0.5161	1	1	0.5	0.3333	0.9744	0.9744	0.1667	0.4	1	1	0.4783
V1+V3+V4	Balance	0.4615	0.9744	0.9744	0.3333	0.6667	0.9474	0.9474	0.4762	0.8235	0.8235	0.8235	0.4571
	Jump	0.7273	0.9302	0.9302	0.5556	0.9302	0.9091	0.9091	0.2	0.8	0.9302	0.9302	0.5313
	Kick	0.766	1	1	0.7778	0.8085	0.9231	0.9231	0.55	0.7692	0.9474	0.9474	0.4828
	Pickup	0.8163	0.9189	0.9189	0.6667	0.9744	0.9474	0.9474	0.2308	0.625	0.9744	0.9744	0.3704
	Punch	0.7407	1	1	0.8182	0.678	0.9756	0.9756	0.5373	0.8696	0.8696	0.8696	0.8
	Stretch	0.7097	0.9756	0.9756	0.8235	0.1818	1	1	0.16	0.7692	1	1	0.7111
V2+V3+V4	Balance	0.3333	0.8235	0.8235	0.2609	0.1739	0.7097	0.7097	0.4615	0.0952	0.6667	0.6667	0.4571
	Jump	0.878	0.8837	0.8837	0.5797	0.9091	0.7917	0.7917	0.15	0.8	0.8444	0.8444	0.5
	Kick	0.625	0.8889	0.8889	0.7429	0.5424	0.8649	0.8649	0.5641	0.6	0.8649	0.8649	0.4444
	Pickup	0.6667	0.9091	0.9091	0.8235	0.6875	0.9756	0.9756	0.4	0.6452	0.9756	0.9756	0.3846
	Punch	0.3871	0.9756	0.9756	0.9	0.6667	0.9302	0.9302	0.5373	0.4211	0.8511	0.8511	0.8
	Stretch	0.6	0.9524	0.9524	0.9744	0.8	1	1	0.16	0.6182	1	1	0.7727
V1+V2+V3+V4	Balance	0.7879	0.9189	0.9189	0.3333	0.2609	0.8571	0.8571	0.4737	0.6667	0.75	0.75	0.4571
	Jump	0.9189	0.8837	0.8837	0.5797	0.8889	0.9048	0.9048	0.1951	1	0.8837	0.8837	0.5075
	Kick	0.6	0.8889	0.8889	0.7647	0.7727	0.8718	0.8718	0.5641	0.5	0.8649	0.8649	0.5
	Pickup	0.7843	0.9524	0.9524	0.8	0.9756	0.9756	0.9756	0.4	0.9756	0.9756	0.9756	0.3846
	Punch	0.8571	1	1	0.9231	0.8163	0.9302	0.9302	0.5373	0.7234	0.8511	0.8511	0.8
	Stretch	0.9189	0.9524	0.9524	0.9744	0.9474	1	1	0.16	0.6667	1	1	0.7727

Table 5.6: Classification Accuracy for Features Vectors with Quantization Unit = 1

	db1				db4				db7			
	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	39.1667	65.8333	65.8333	66.6667	39.1667	65.8333	65.8333	66.6667	39.1667	65.8333	65.8333	66.6667
Euc	40.8333	22.5	22.5	35	40.8333	22.5	22.5	35	40.8333	22.5	22.5	35
DWTcAStat	65	77.5	77.5	43.3333	76.6667	81.6667	81.6667	40.8333	68.3333	70.8333	70.8333	42.5
DWTcDStat	38.3333	21.6667	21.6667	42.5	29.1667	30	30	43.3333	36.6667	25.8333	25.8333	38.3333
V1+V2	50.8333	42.5	42.5	40.8333	50.8333	42.5	42.5	40.8333	50.8333	42.5	42.5	40.8333
V1+V3	69.1667	88.3333	88.3333	43.3333	60.8333	88.3333	88.3333	42.5	64.1667	75	75	42.5
V1+V4	61.6667	37.5	37.5	42.5	40.8333	43.3333	43.3333	43.3333	39.1667	36.6667	36.6667	38.3333
V2+V3	57.5	55.8333	55.8333	45	62.5	58.3333	58.3333	43.3333	58.3333	52.5	52.5	41.6667
V2+V4	45	25	25	43.3333	40	34.1667	34.1667	41.6667	40	31.6667	31.6667	38.3333
V3+V4	73.3333	68.3333	68.3333	36.6667	68.3333	55.8333	55.8333	47.5	70.8333	50	50	46.6667
V1+V2+V3	70.8333	64.1667	64.1667	45.8333	65	65.8333	65.8333	44.1667	64.1667	60	60	41.6667
V1+V2+V4	56.6667	36.6667	36.6667	44.1667	52.5	38.3333	38.3333	41.6667	44.1667	37.5	37.5	39.1667
V1+V3+V4	65	73.33333	73.33333	36.6667	66.66667	65	65	47.5	62.5	60.83333	60.83333	46.6667
V2+V3+V4	69.1667	58.3333	58.3333	36.6667	68.3333	54.1667	54.1667	48.3333	70	52.5	52.5	47.5
V1+V2+V3+V4	78.3333	65	65	35.8333	62.5	61.6667	61.6667	49.1667	62.5	58.3333	58.3333	47.5

Table 5.7: Classification F-Measure for Features Vectors with Quantization Unit = 1

		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	Balance	0.2	0.727	0.727	0.833	0.2	0.727	0.727	0.833	0.2	0.727	0.727	0.833
	Jump	0	0.615	0.615	0.069	0	0.615	0.615	0.069	0	0.615	0.615	0.069
	Kick	0.1	0.293	0.293	0.533	0.1	0.293	0.293	0.533	0.1	0.293	0.293	0.533
	Pickup	0.4	0.711	0.711	0.559	0.4	0.711	0.711	0.559	0.4	0.711	0.711	0.559
	Punch	0.6	0.889	0.889	0.974	0.6	0.889	0.889	0.974	0.6	0.889	0.889	0.974
	Stretch	0.8	0.788	0.788	0.947	0.8	0.788	0.788	0.947	0.8	0.788	0.788	0.947
Euc	Balance	0.4	0.308	0.308	0.292	0.4	0.308	0.308	0.292	0.4	0.308	0.308	0.292
	Jump	0.3	0.054	0.054	0.091	0.3	0.054	0.054	0.091	0.3	0.054	0.054	0.091
	Kick	0.4	0	0	0.333	0.4	0	0	0.333	0.4	0	0	0.333
	Pickup	0.5	0.278	0.278	0.529	0.5	0.278	0.278	0.529	0.5	0.278	0.278	0.529
	Punch	0.3	0.08	0.08	0	0.3	0.08	0.08	0	0.3	0.08	0.08	0
	Stretch	0.5	0.389	0.389	0.655	0.5	0.389	0.389	0.655	0.5	0.389	0.389	0.655
DWTcAStat	Balance	0.7	0.75	0.75	0	0.7	0.75	0.75	0	0.6	0.571	0.571	0
	Jump	0	0.345	0.345	0	0.5	0.588	0.588	0	0	0	0	0
	Kick	0.6	0.889	0.889	0.625	0.9	0.842	0.842	0.5	0.8	0.919	0.919	0.385
	Pickup	0.5	0.635	0.635	0.423	0.6	0.714	0.714	0.429	0.6	0.563	0.563	0.447
	Punch	1	1	1	0.571	1	1	1	0.579	1	1	1	0.718
	Stretch	1	1	1	0.872	1	1	1	0.821	1	0.976	0.976	0.769
DWTcDStat	Balance	0.3	0.222	0.222	0.508	0.1	0.182	0.182	0.324	0.3	0.095	0.095	0
	Jump	0.2	0.163	0.163	0.121	0	0.091	0.091	0.557	0.1	0	0	0.52
	Kick	0.5	0.329	0.329	0.773	0.3	0.444	0.444	0.566	0.3	0.377	0.377	0.557
	Pickup	0.4	0.158	0.158	0	0.4	0.4	0.4	0.182	0.4	0.385	0.385	0.333
	Punch	0.6	0.095	0.095	0.533	0.7	0	0	0.511	0.9	0	0	0.303
	Stretch	0.3	0.154	0.154	0	0.1	0	0	0	0.1	0	0	0
V1+V2	Balance	0.3	0.541	0.541	0.4	0.3	0.541	0.541	0.4	0.3	0.541	0.541	0.4
	Jump	0.2	0.238	0.238	0.13	0.2	0.238	0.238	0.13	0.2	0.238	0.238	0.13
	Kick	0.4	0.195	0.195	0.389	0.4	0.195	0.195	0.389	0.4	0.195	0.195	0.389
	Pickup	0.4	0.426	0.426	0.541	0.4	0.426	0.426	0.541	0.4	0.426	0.426	0.541
	Punch	0.7	0.4	0.4	0	0.7	0.4	0.4	0	0.7	0.4	0.4	0
	Stretch	0.8	0.708	0.708	0.755	0.8	0.708	0.708	0.755	0.8	0.708	0.708	0.755
V1+V3	Balance	0.9	0.947	0.947	0	0.5	0.889	0.889	0	0.5	0.71	0.71	0
	Jump	0	0.684	0.684	0	0	0.732	0.732	0	0	0.222	0.222	0
	Kick	0.6	0.857	0.857	0.625	0.5	0.857	0.857	0.6	0.7	0.889	0.889	0.385
	Pickup	0.6	0.816	0.816	0.423	0.5	0.833	0.833	0.435	0.5	0.606	0.606	0.447
	Punch	1	1	1	0.571	1	1	1	0.579	1	1	1	0.718
	Stretch	1	1	1	0.872	1	1	1	0.821	1	1	1	0.769
V1+V4	Balance	0.5	0.37	0.37	0.508	0	0.333	0.333	0.324	0.1	0.4	0.4	0
	Jump	0.5	0.245	0.245	0.121	0.1	0.6	0.6	0.557	0	0.133	0.133	0.52
	Kick	0.3	0.313	0.313	0.773	0.1	0.432	0.432	0.566	0.3	0.41	0.41	0.557
	Pickup	0.6	0.455	0.455	0	0.4	0.576	0.576	0.182	0.4	0.623	0.623	0.333
	Punch	1	0.645	0.645	0.533	0.8	0	0	0.511	0.9	0	0	0.303
	Stretch	0.8	0.32	0.32	0	0.8	0.261	0.261	0	0.4	0.095	0.095	0
V2+V3	Balance	0.5	0.444	0.444	0	0.5	0.462	0.462	0	0.4	0.261	0.261	0
	Jump	0.1	0.182	0.182	0	0.2	0.278	0.278	0.057	0.2	0.129	0.129	0
	Kick	0.7	0.686	0.686	0.667	0.8	0.684	0.684	0.571	0.8	0.765	0.765	0.444
	Pickup	0.7	0.557	0.557	0.464	0.7	0.597	0.597	0.472	0.7	0.492	0.492	0.453
	Punch	0.6	0.581	0.581	0.541	0.8	0.581	0.581	0.5	0.6	0.581	0.581	0.649
	Stretch	0.6	0.755	0.755	0.872	0.7	0.769	0.769	0.85	0.6	0.714	0.714	0.714

		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
V2+V4	Balance	0.3	0.222	0.222	0.523	0.3	0.167	0.167	0.359	0.5	0.091	0.091	0.087
	Jump	0.2	0.133	0.133	0.182	0.2	0.186	0.186	0.515	0	0.25	0.25	0.5
	Kick	0.5	0.333	0.333	0.773	0.5	0.475	0.475	0.549	0.4	0.515	0.515	0.54
	Pickup	0.6	0.188	0.188	0	0.5	0.436	0.436	0.182	0.5	0.345	0.345	0.333
	Punch	0.4	0.091	0.091	0.517	0.5	0.091	0.091	0.476	0.5	0	0	0.323
	Stretch	0.6	0.381	0.381	0	0.5	0.432	0.432	0	0.3	0.294	0.294	0
V3+V4	Balance	0.8	0.71	0.71	0.148	0.4	0.519	0.519	0.286	0.6	0.261	0.261	0
	Jump	0.2	0.323	0.323	0.14	0.4	0.182	0.182	0.525	0	0	0	0.632
	Kick	0.9	0.667	0.667	0.809	0.8	0.93	0.93	0.667	0.9	0.952	0.952	0.667
	Pickup	0.6	0.714	0.714	0.333	0.6	0.444	0.444	0.333	0.5	0.408	0.408	0.4
	Punch	1	0.75	0.75	0.405	1	0.095	0.095	0.577	1	0.095	0.095	0.491
	Stretch	1	0.872	0.872	0	0.8	0.919	0.919	0	1	0.889	0.889	0
V1+V2+V3	Balance	0.9	0.6	0.6	0	0.5	0.6	0.6	0.067	0.6	0.462	0.462	0
	Jump	0	0.27	0.27	0	0.1	0.333	0.333	0.057	0.1	0.194	0.194	0
	Kick	0.7	0.647	0.647	0.688	0.8	0.686	0.686	0.571	0.7	0.727	0.727	0.444
	Pickup	0.6	0.625	0.625	0.479	0.7	0.615	0.615	0.479	0.6	0.533	0.533	0.453
	Punch	0.9	0.727	0.727	0.556	0.9	0.727	0.727	0.5	0.9	0.688	0.688	0.649
	Stretch	1	0.952	0.952	0.872	0.8	0.976	0.976	0.85	0.8	0.93	0.93	0.714
V1+V2+V4	Balance	0.6	0.429	0.429	0.54	0.4	0.4	0.4	0.359	0.4	0.261	0.261	0.087
	Jump	0.4	0.267	0.267	0.177	0.4	0.273	0.273	0.515	0	0.316	0.316	0.5
	Kick	0.6	0.333	0.333	0.773	0.3	0.351	0.351	0.549	0.4	0.419	0.419	0.54
	Pickup	0.5	0.308	0.308	0	0.5	0.467	0.467	0.182	0.5	0.413	0.413	0.4
	Punch	0.7	0.4	0.4	0.542	0.7	0.095	0.095	0.476	0.7	0.095	0.095	0.323
	Stretch	0.6	0.541	0.541	0	0.7	0.606	0.606	0	0.6	0.546	0.546	0
V1+V3+V4	Balance	0.6	0.824	0.824	0.148	0.5	0.667	0.667	0.286	0.5	0.519	0.519	0
	Jump	0.2	0.412	0.412	0.14	0.3	0.333	0.333	0.525	0.1	0.083	0.083	0.632
	Kick	0.6	0.667	0.667	0.809	0.8	0.8	0.8	0.667	0.6	0.85	0.85	0.667
	Pickup	0.6	0.72	0.72	0.333	0.6	0.548	0.548	0.333	0.5	0.494	0.494	0.4
	Punch	1	0.889	0.889	0.405	1	0.621	0.621	0.577	0.9	0.667	0.667	0.491
	Stretch	1	0.895	0.895	0	0.9	0.947	0.947	0	1	0.947	0.947	0
V2+V3+V4	Balance	0.6	0.519	0.519	0.214	0.8	0.4	0.4	0.4	0.6	0.182	0.182	0
	Jump	0.3	0.229	0.229	0.136	0.2	0.125	0.125	0.516	0.5	0.133	0.133	0.632
	Kick	0.8	0.588	0.588	0.766	0.8	0.667	0.667	0.667	0.8	0.85	0.85	0.667
	Pickup	0.8	0.642	0.642	0.333	0.7	0.537	0.537	0.333	0.6	0.487	0.487	0.462
	Punch	0.7	0.5	0.5	0.416	0.7	0.32	0.32	0.571	0.9	0.333	0.333	0.5
	Stretch	0.7	0.87	0.87	0	0.8	0.93	0.93	0	0.7	0.826	0.826	0
V1+V2+V3+V4	Balance	0.6	0.552	0.552	0.148	0.5	0.571	0.571	0.4	0.5	0.333	0.333	0
	Jump	0.4	0.343	0.343	0.133	0	0.286	0.286	0.525	0	0.2	0.2	0.632
	Kick	0.9	0.609	0.609	0.766	0.6	0.651	0.651	0.667	0.8	0.821	0.821	0.667
	Pickup	0.8	0.704	0.704	0.333	0.6	0.606	0.606	0.333	0.6	0.52	0.52	0.462
	Punch	1	0.71	0.71	0.416	1	0.519	0.519	0.6	0.9	0.519	0.519	0.5
	Stretch	0.9	0.889	0.889	0	0.8	0.976	0.976	0	0.8	0.93	0.93	0

Table 5.8: Classification Accuracy for Features Vectors with Quantization Unit = 0.5

	db1				db4				db7			
	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	70	68.3333	68.3333	82.5	70	68.3333	68.3333	82.5	70	68.3333	68.3333	82.5
Euc	41.6667	40.8333	40.8333	60.8333	41.6667	40.8333	40.8333	60.8333	41.6667	40.8333	40.8333	60.8333
DWTcAStat	89.1667	86.6667	86.6667	79.1667	91.6667	92.5	92.5	72.5	88.3333	95	95	68.3333
DWTcDStat	32.5	26.6667	26.6667	45.8333	40.8333	30.8333	30.8333	42.5	31.6667	30	30	27.5
V1+V2	83.3333	61.6667	61.6667	71.6667	83.3333	61.6667	61.6667	71.6667	83.3333	61.6667	61.6667	71.6667
V1+V3	84.1667	90	90	80	88.3333	91.6667	91.6667	72.5	78.3333	94.1667	94.1667	70
V1+V4	71.6667	38.3333	38.3333	45.8333	61.6667	50	50	43.3333	71.6667	50	50	27.5
V2+V3	86.6667	78.3333	78.3333	81.6667	80.8333	82.5	82.5	70.8333	90	81.6667	81.6667	70
V2+V4	55.8333	36.6667	36.6667	45.8333	49.1667	40.8333	40.8333	43.3333	36.6667	37.5	37.5	26.6667
V3+V4	94.1667	64.1667	64.1667	54.1667	94.1667	86.6667	86.6667	52.5	93.3333	80.8333	80.8333	32.5
V1+V2+V3	91.6667	86.6667	86.6667	82.5	90.8333	90	90	70.8333	90	89.1667	89.1667	70.8333
V1+V2+V4	77.5	50.8333	50.8333	45.8333	70	58.3333	58.3333	44.1667	75	54.1667	54.1667	26.6667
V1+V3+V4	98.3333	81.6667	81.6667	54.1667	91.6667	89.1667	89.1667	53.3333	92.5	88.3333	88.3333	32.5
V2+V3+V4	92.5	67.5	67.5	55	88.3333	77.5	77.5	54.1667	95.8333	70	70	32.5
V1+V2+V3+V4	85	79.1667	79.1667	55	84.1667	90	90	54.1667	94.1667	87.5	87.5	33.3333

Table 5.9: Classification F-Measure for Features Vectors with Quantization Unit = 0.5

		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
Stdev	Balance	0.6	0.737	0.737	0.865	0.6	0.737	0.737	0.865	0.6	0.737	0.737	0.865
	Jump	0.8	0.816	0.816	0.75	0.8	0.816	0.816	0.75	0.8	0.816	0.816	0.75
	Kick	0.8	0.7	0.7	0.621	0.8	0.7	0.7	0.621	0.8	0.7	0.7	0.621
	Pickup	0.4	0.565	0.565	0.889	0.4	0.565	0.565	0.889	0.4	0.565	0.565	0.889
	Punch	0.9	0.737	0.737	0.844	0.9	0.737	0.737	0.844	0.9	0.737	0.737	0.844
	Stretch	0.6	0.483	0.483	0.909	0.6	0.483	0.483	0.909	0.6	0.483	0.483	0.909
Euc	Balance	0.2	0.207	0.207	0.667	0.2	0.207	0.207	0.667	0.2	0.207	0.207	0.667
	Jump	0.3	0.25	0.25	0.419	0.3	0.25	0.25	0.419	0.3	0.25	0.25	0.419
	Kick	0.4	0.298	0.298	0.444	0.4	0.298	0.298	0.444	0.4	0.298	0.298	0.444
	Pickup	0.5	0.526	0.526	0.743	0.5	0.526	0.526	0.743	0.5	0.526	0.526	0.743
	Punch	0.1	0.167	0.167	0.48	0.1	0.167	0.167	0.48	0.1	0.167	0.167	0.48
	Stretch	0.8	0.872	0.872	0.93	0.8	0.872	0.872	0.93	0.8	0.872	0.872	0.93
DWTcAStat	Balance	0.9	0.919	0.919	0.784	1	1	1	0.645	0.9	0.974	0.974	0.615
	Jump	0.8	0.727	0.727	0.524	0.8	0.816	0.816	0.683	0.7	0.884	0.884	0.526
	Kick	0.9	0.889	0.889	0.75	0.9	0.947	0.947	0.722	0.9	1	1	0.811
	Pickup	0.8	0.71	0.71	0.788	0.8	0.788	0.788	0.385	0.8	0.865	0.865	0.231
	Punch	1	0.976	0.976	0.974	1	1	1	0.857	1	0.976	0.976	0.889
	Stretch	1	1	1	0.93	1	1	1	1	1	1	1	0.947
DWTcDStat	Balance	0	0	0	0.438	0.3	0	0	0.083	0	0	0	0.174
	Jump	0.4	0.36	0.36	0.588	0.8	0.652	0.652	0.741	0.9	0.615	0.615	0.388
	Kick	0.6	0.5	0.5	0.593	0.4	0.381	0.381	0.717	0.3	0.37	0.37	0.457
	Pickup	0.1	0	0	0	0.4	0.138	0.138	0	0.1	0	0	0.091
	Punch	0.4	0	0	0.546	0.1	0	0	0.319	0	0	0	0.108
	Stretch	0	0	0	0	0	0	0	0	0	0	0	0
V1+V2	Balance	0.7	0.5	0.5	0.811	0.7	0.5	0.5	0.811	0.7	0.5	0.5	0.811
	Jump	0.9	0.581	0.581	0.667	0.9	0.581	0.581	0.667	0.9	0.581	0.581	0.667
	Kick	0.8	0.524	0.524	0.444	0.8	0.524	0.524	0.444	0.8	0.524	0.524	0.444
	Pickup	0.8	0.593	0.593	0.743	0.8	0.593	0.593	0.743	0.8	0.593	0.593	0.743
	Punch	0.9	0.522	0.522	0.681	0.9	0.522	0.522	0.681	0.9	0.522	0.522	0.681
	Stretch	0.8	0.974	0.974	0.93	0.8	0.974	0.974	0.93	0.8	0.974	0.974	0.93
V1+V3	Balance	0.9	1	1	0.8	0.9	1	1	0.645	0.7	1	1	0.645
	Jump	0.7	0.769	0.769	0.524	0.7	0.8	0.8	0.683	0.7	0.844	0.844	0.564
	Kick	1	0.919	0.919	0.75	0.9	0.947	0.947	0.722	0.8	0.974	0.974	0.79
	Pickup	0.5	0.71	0.71	0.824	0.8	0.75	0.75	0.385	0.4	0.833	0.833	0.231
	Punch	1	1	1	0.974	1	1	1	0.857	1	1	1	0.889
	Stretch	1	1	1	0.93	1	1	1	1	1	1	1	0.974
V1+V4	Balance	0.7	0.462	0.462	0.438	0.6	0.727	0.727	0.16	0.6	0.571	0.571	0.174
	Jump	0.7	0.471	0.471	0.588	0.6	0.702	0.702	0.741	0.8	0.667	0.667	0.388
	Kick	0.8	0.54	0.54	0.593	0.8	0.548	0.548	0.717	0.6	0.506	0.506	0.457
	Pickup	0.2	0.091	0.091	0	0.5	0.364	0.364	0	0.5	0.6	0.6	0.091
	Punch	1	0	0	0.546	0.8	0.095	0.095	0.324	0.9	0	0	0.108
	Stretch	0.8	0.167	0.167	0	0.4	0.087	0.087	0	0.8	0.261	0.261	0
V2+V3	Balance	0.9	0.621	0.621	0.851	0.7	0.667	0.667	0.625	1	0.71	0.71	0.625
	Jump	0.6	0.591	0.591	0.651	0.8	0.718	0.718	0.683	0.9	0.629	0.629	0.6
	Kick	1	0.778	0.778	0.552	0.8	0.872	0.872	0.563	0.9	0.9	0.9	0.743
	Pickup	0.8	0.8	0.8	0.85	0.8	0.8	0.8	0.385	0.8	0.735	0.735	0.24
	Punch	0.9	0.889	0.889	0.974	0.9	0.87	0.87	0.919	0.9	0.909	0.909	0.889
	Stretch	0.9	0.976	0.976	0.952	0.9	0.976	0.976	1	0.9	0.976	0.976	1

		db1				db4				db7			
Features Vector	Actions	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM	RF	3NN	5NN	SVM
V2+V4	Balance	0.3	0	0	0.438	0.2	0.095	0.095	0.16	0.3	0	0	0.091
	Jump	0.7	0.594	0.594	0.588	0.8	0.842	0.842	0.755	0.7	0.682	0.682	0.388
	Kick	0.5	0.329	0.329	0.593	0.4	0.383	0.383	0.717	0.4	0.344	0.344	0.457
	Pickup	0.6	0.571	0.571	0	0.6	0.512	0.512	0	0.5	0.619	0.619	0.091
	Punch	0.2	0	0	0.546	0	0	0	0.319	0	0	0	0.105
	Stretch	0.9	0.261	0.261	0	0.8	0.25	0.25	0	0	0.095	0.095	0
V3+V4	Balance	0.9	0.333	0.333	0.571	0.9	0.857	0.857	0.438	1	0.571	0.571	0.174
	Jump	0.9	0.482	0.482	0.656	0.9	0.741	0.741	0.741	0.8	0.635	0.635	0.412
	Kick	0.9	0.974	0.974	0.691	1	0.976	0.976	0.818	0.9	1	1	0.452
	Pickup	0.9	0.095	0.095	0	0.9	0.667	0.667	0	0.9	0.621	0.621	0.095
	Punch	1	0.824	0.824	0.667	1	0.974	0.974	0.514	1	1	1	0.348
	Stretch	1	0.974	0.974	0	1	0.976	0.976	0	1	1	1	0.091
V1+V2+V3	Balance	1	0.788	0.788	0.851	0.9	0.889	0.889	0.625	0.9	0.919	0.919	0.635
	Jump	0.8	0.79	0.79	0.651	0.8	0.811	0.811	0.683	0.8	0.743	0.743	0.6
	Kick	1	0.889	0.889	0.571	0.9	0.919	0.919	0.563	0.9	0.919	0.919	0.743
	Pickup	0.9	0.87	0.87	0.878	0.9	0.87	0.87	0.385	0.9	0.833	0.833	0.308
	Punch	1	0.87	0.87	0.974	1	0.93	0.93	0.919	1	0.952	0.952	0.889
	Stretch	1	0.976	0.976	0.952	1	0.976	0.976	1	0.9	0.976	0.976	1
V1+V2+V4	Balance	0.8	0.519	0.519	0.438	0.5	0.645	0.645	0.16	0.8	0.462	0.462	0.091
	Jump	0.7	0.645	0.645	0.588	0.8	0.923	0.923	0.769	0.7	0.821	0.821	0.388
	Kick	0.7	0.444	0.444	0.593	0.6	0.48	0.48	0.741	0.7	0.456	0.456	0.457
	Pickup	0.7	0.6	0.6	0	0.8	0.558	0.558	0	0.8	0.622	0.622	0.091
	Punch	0.9	0.182	0.182	0.546	0.9	0.182	0.182	0.319	0.9	0.261	0.261	0.105
	Stretch	0.9	0.519	0.519	0	0.7	0.667	0.667	0	0.7	0.571	0.571	0
V1+V3+V4	Balance	1	0.919	0.919	0.571	0.9	0.947	0.947	0.485	0.8	0.919	0.919	0.174
	Jump	1	0.656	0.656	0.656	0.9	0.8	0.8	0.741	1	0.8	0.8	0.412
	Kick	1	0.947	0.947	0.691	0.9	0.952	0.952	0.818	0.9	0.952	0.952	0.452
	Pickup	1	0.462	0.462	0	0.8	0.71	0.71	0	1	0.71	0.71	0.095
	Punch	1	0.974	0.974	0.667	1	0.947	0.947	0.522	1	0.947	0.947	0.348
	Stretch	1	0.923	0.923	0	0.9	0.976	0.976	0	0.9	0.952	0.952	0.091
V2+V3+V4	Balance	0.8	0.333	0.333	0.556	0.9	0.519	0.519	0.438	1	0.261	0.261	0.174
	Jump	1	0.58	0.58	0.656	0.9	0.851	0.851	0.769	1	0.667	0.667	0.412
	Kick	0.9	0.723	0.723	0.691	0.7	0.741	0.741	0.864	0.9	0.679	0.679	0.452
	Pickup	0.9	0.75	0.75	0	1	0.829	0.829	0	0.9	0.79	0.79	0.095
	Punch	0.9	0.571	0.571	0.681	0.8	0.621	0.621	0.528	1	0.621	0.621	0.348
	Stretch	1	1	1	0.095	1	0.952	0.952	0	1	1	1	0.091
V1+V2+V3+V4	Balance	0.8	0.71	0.71	0.556	0.8	0.889	0.889	0.438	1	0.889	0.889	0.174
	Jump	0.7	0.714	0.714	0.656	0.7	0.976	0.976	0.769	0.8	0.905	0.905	0.417
	Kick	0.9	0.727	0.727	0.691	0.9	0.851	0.851	0.864	1	0.816	0.816	0.5
	Pickup	0.7	0.8	0.8	0	0.8	0.85	0.85	0	0.9	0.842	0.842	0.095
	Punch	1	0.824	0.824	0.681	1	0.857	0.857	0.528	1	0.824	0.824	0.348
	Stretch	1	1	1	0.095	0.9	0.976	0.976	0	1	0.976	0.976	0.091

5.3 Features Importance

In the previous sections, we extracted four types of features to recognize the human action. Here, we used features selection technique (i.e., forward features selection) to evaluate the significance of the extracted features. Therefore, we selected two feattures vectors that have the highest scores in each experimental design:

- Experimental design 1
  - Features vector: V1+V2+V3+V4 using db1
  - Score: 99.2857
  - Classifier: SVM
- Experimental design 2
  - Features vector: V1+V3+V4 using db1 and half unit quantization
  - Score: 98.3333
  - Classifier: RF

The features selected based on the training data using 10 folds cross validation. When they were tested on test data with unseen subjects the performance

Table 5.10: Classification before and after forward features selection

Experimental Design 1				
V1+V2+V3+V4	*Before Features Selection*		*After Features Selection*	
	Number of predictors	SVM accuracy	Number of predictors	SVM accuracy
	297	99.2857	50	81.4286
Experimental Design 2				
V1+V3+V4	*Before Features Selection*		*After Features Selection*	
	Number of predictors	RF accuracy	Number of predictors	RF accuracy
	237	98.3333	23	73.3333

drops (as shown in Table 5.10) due to lack of generalization. This indicates the importance of all features especially when testing on unseen subjects.

5.4 Threats to validity

Many threats can affect the validity of any experimental design. Conclusion validity, construct validity, internal validity, and external validity, are the four sides of the experimental design overall validity.

In our experimental design, the quantity of the human action data can be a threat to the conclusion validity. In order to minimize the impact of this threat, we used data collected from three different sources (Ibañez et al., AutoDesk Maya, our Kinect). This data has 12 subjects and 13 actions. Moreover, the bias or unexpected behavior of the learning algorithm can be a threat to the construction validity. Four learning algorithms (RF, 3NN, 5NN, SVM) used in the experiments to mitigate this threat.

The cause of classification performance drops can be an internal threat to validity. This justified our choice to use four learning algorithms, such that if all have the same performance drop it clearly indicates that the features were not sufficient. Finally, the generalization of our results to a wide range of human actions is an external threat to validity. We tried to reduce this threat by using 13 actions performed by upper and lower body parts.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Summary and Results

Human action recognition is one of the important aspects of computer vision field. It contributes to vital applications such as security, health, and video categorization applications. Microsoft Kinect was a huge boost to the studies in this field. The joints tracking system provided by Kinect helped researchers to model complex human actions instead of using traditional sources such as RGB and depth 2D videos.

Since the Kinect joints tracking system is an invariant representation of the human body, many studies used the raw positions of the human joints as features to train the learning algorithm. Nevertheless, the raw joints positions might not be the best descriptors for the human actions, because they are vulnerable to

variations in the subject behavior while performing the action.

We noticed in the literature that extracting new features from the raw joints positions was an area for improvement. Therefore, in our first experimental design we addressed this issue and extract new features from raw joints positions. The experimental findings indicate an improved recognition rates after extracting the features compared to the performance of raw joints positions as features.

Another problem related to the human action recognition, is the need of adequate, comprehensive, and diverse training dataset. Many studies used real human subjects to record actions using Kinect, and then use them for training. However, this process is time and effort consuming, because the human subject needs a full understanding about the actions, which involves communication and explanation sessions. On the other hand, computer 3D graphics provides an excellent simulation of the human subjects. Therefore, our second experimental design was dedicated to investigate the feasibility of training the human action recognition classifiers on synthetic human action data then used real human action data for testing. Our experiments indicate that the classifiers that trained on synthetic data were able to recognize real human actions.

6.2 Future Work

In this thesis, we addressed two issues related to human action recognition. First, we proposed new features to characterize the human actions defined in Section 4.3. Second, we generate large synthetic dataset to provide comprehensive training for

human action classifier.

Future work can encompasses checking whether the proposed features are the best features to describe other actions. Furthermore, it is interesting to explore the existence of a common set of features that can be useful to recognize in-place human actions in general, not only the ones used in this study.

On the other hand, the process of generating synthetic data can be extended into different directions. For instance, one can create a real time pipeline that captures the action of a real subject and produces different variations out of it synthetically (i.e., different body builds or different viewing angles). Furthermore, in the current implementation we applied manual methods of segmentation, since real time segmentation is beyond the scope of this study. It is desirable to merge real time segmentation with the learning process.

Appendices

APPENDIX A

GENERATE AND PREPROCESS REAL HUMAN ACTION DATA

A.1 Record Human Action Data using Kinect

To record real human action data, you need to have:

- Kinect for windows v2.
- Kinect SDK 2.0.
- C# editor (Visual Studio 2013 for Desktop was used in this tutorial).

The code shown in Listing A.1 is a modification of the project (Color Basics-WPF) which is bundled with Kinect SDK 2.0 browser. The objective of the original project is to display a real time color image using Kinect. It was modified to perform the following tasks:

- Collect depth data using DepthFrame stream.
- Collect joints world positions using BodyFrame stream.
- Subtract the background from depth data using BodyIndexFrame stream.
- Write the depth data and the joints positions to a directory specified by the user.

This project can record up to 1500 frames of depth and joints positions data (see line 47 and 58 of Listing A.1). The collection was done in a pattern of keeping one frame and ignoring the following one (see line 310 of Listing A.1). We justified that, by saying there are no much information in two consecutive frames and to save space. Also, The data was collected only if a human appears in the scene (see line 315 of Listing A.1). The collected 15 joints are (see lines 333 to 427 of Listing A.1): Head, Neck, SpineMid, ShoulderLeft, ElbowLeft, HandLeft, ShoulderRight,

ElbowRight, HandRight, HipLeft, KneeLeft, FootLeft, HipRight, KneeRight, and FootRight. The background subtraction was done by nullifying each pixel in DepthFrame that does not have a player index in BodyIndexFrame (see line 239 to 246 of Listing A.1).
This project should produce:

- 1500 text files containing the depth data of 1500 frames. If the frame has no information the file will be filled by zeros.
- A single text file of 1500 lines where each line corresponds to a frame. Each line has 45 columns which are the world positions (x, y, x) of aforementioned joints.

Listing A.1: Code to record joints positions and depth frames using Kinect SDK

```
1   //
    ------------------------------------------------------------------------------

2  // <copyright file="MainWindow.xaml.cs" company="Microsoft">
3  //     Copyright (c) Microsoft Corporation.  All rights reserved.
4  // </copyright>
5  //------------------------------------------------------------------------------
6  namespace Microsoft.Samples.Kinect.ColorBasics
7  {
8      using System;
9      using System.Collections.Generic;
10     using System.ComponentModel;
11     using System.Diagnostics;
12     using System.Globalization;
13     using System.IO;
14     using System.Windows;
15     using System.Windows.Media;
16     using System.Windows.Media.Imaging;
17     using Microsoft.Kinect;
18
19     /// <summary>
20     /// Interaction logic for MainWindow
21     /// </summary>
22     public partial class MainWindow : Window, INotifyPropertyChanged
23     {
24         /// <summary>
25         /// Active Kinect sensor
26         /// </summary>
27         private KinectSensor kinectSensor = null;
28
29         /// <summary>
30         /// Coordinate mapper to map one type of point to another
31         /// </summary>
32         private CoordinateMapper coordinateMapper = null;
33
34         /// <summary>
35         /// Reader for color frames
36         /// </summary>
37         private MultiSourceFrameReader multiReader = null;
38
39         /// <summary>
40         /// Bitmap to display
41         /// </summary>
42         private WriteableBitmap colorBitmap = null;
43
44         /// <summary>
45         /// Maximum frames to record
```

```
46          /// </summary>
47          private int frameMax = 1500;
48
49          /// <summary>
50          /// Holds depth data
51          /// Holds body index data
52          /// Holds joints coordinates rows are frames, columns are coordinates
53          /// </summary>
54          ushort[,] depthData = null;
55          ushort[] tempDepthData = null;
56          byte[,] bodyIndexData = null;
57          byte[] tempBodyIndexData = null;
58          private double[,] coordArray = new double[1500, 45];
59
60          /// <summary>
61          /// Holds depth data
62          /// </summary>
63
64
65          /// <summary>
66          /// counts coming frames
67          /// </summary>
68          private int frameCounter;
69          private int depthDataCounter1 = 0;
70          private int recordCounter = 0;
71          private int timeStep;
72          private int coordDataCounter1 = 0;
73
74          /// <summary>
75          /// Current status text to display
76          /// </summary>
77          private string statusText = null;
78
79          /// <summary>
80          /// Array for the bodies
81          /// </summary>
82          private Body[] bodies = null;
83
84          /// <summary>
85          /// Description of the data contained in the depth frame
86          /// Description of the data contained in the body index frame
87          /// </summary>
88          private FrameDescription depthFrameDescription = null;
89          private FrameDescription bodyIndexFrameDescription = null;
90
91          /// <summary>
92          /// Constant for clamping Z values of camera space points from being
                negative
93          /// </summary>
94          private const float InferredZPositionClamp = 0.1f;
95
96          /// <summary>
97          /// Initializes a new instance of the MainWindow class.
98          /// </summary>
99          public MainWindow()
100         {
101             // get the kinectSensor object
102             this.kinectSensor = KinectSensor.GetDefault();
103
104             // open the reader for the color frames
105             this.multiReader = this.kinectSensor.OpenMultiSourceFrameReader(
                    FrameSourceTypes.Color |
106                                             FrameSourceTypes.Depth |
                                                FrameSourceTypes.BodyIndex |
107                                             FrameSourceTypes.Body);
108
109             // wire handler for frame arrival
```

```
110             this.multiReader.MultiSourceFrameArrived += this.
                    Reader_MultiSourceFrameArrived;
111
112             // create the colorFrameDescription from the ColorFrameSource using
                    Bgra format
113             FrameDescription colorFrameDescription = this.kinectSensor.
                    ColorFrameSource.CreateFrameDescription(ColorImageFormat.Bgra);
114             this.depthFrameDescription = this.kinectSensor.DepthFrameSource.
                    FrameDescription;
115             this.bodyIndexFrameDescription = this.kinectSensor.
                    BodyIndexFrameSource.FrameDescription;
116
117             // create the bitmap to display
118             this.colorBitmap = new WriteableBitmap(colorFrameDescription.Width,
                    colorFrameDescription.Height, 96.0, 96.0, PixelFormats.Bgr32,
                    null);
119
120             // set IsAvailableChanged event notifier
121             this.kinectSensor.IsAvailableChanged += this.
                    Sensor_IsAvailableChanged;
122
123             // open the sensor
124             this.kinectSensor.Open();
125
126             // set the status text
127             this.StatusText = this.kinectSensor.IsAvailable ? Properties.
                    Resources.RunningStatusText
128                                                             : Properties.
                                                                Resources.
                                                                NoSensorStatusText
                                                                ;
129
130             // use the window object as the view model in this simple example
131             this.DataContext = this;
132
133             // initialize the components (controls) of the window
134             this.InitializeComponent();
135
136             // initialize some arrays based on the depth frame size
137
138             // Depth frame (512x424)
139             int depthWidth = this.depthFrameDescription.Width;
140             int depthHeight = this.depthFrameDescription.Height;
141
142             // Body index frame (512x424)
143             int bodyIndexWidth = this.bodyIndexFrameDescription.Width;
144             int bodyIndexHeight = this.bodyIndexFrameDescription.Height;
145
146             this.depthData = new ushort[frameMax, depthWidth * depthHeight];
147             this.tempDepthData = new ushort[depthWidth * depthHeight];
148
149             this.bodyIndexData = new byte[frameMax, bodyIndexWidth *
                    bodyIndexHeight];
150             this.tempBodyIndexData = new byte[bodyIndexWidth * bodyIndexHeight];
151         }
152
153         /// <summary>
154         /// INotifyPropertyChangedPropertyChanged event to allow window controls
                to bind to changeable data
155         /// </summary>
156         public event PropertyChangedEventHandler PropertyChanged;
157
158         /// <summary>
159         /// Gets the bitmap to display
160         /// </summary>
161         public ImageSource ImageSource
162         {
163             get
```

```
164             {
165                 return this.colorBitmap;
166             }
167         }
168 
169         /// <summary>
170         /// Gets or sets the current status text to display
171         /// </summary>
172         public string StatusText
173         {
174             get
175             {
176                 return this.statusText;
177             }
178 
179             set
180             {
181                 if (this.statusText != value)
182                 {
183                     this.statusText = value;
184 
185                     // notify any bound elements that the text has changed
186                     if (this.PropertyChanged != null)
187                     {
188                         this.PropertyChanged(this, new PropertyChangedEventArgs("
                            StatusText"));
189                     }
190                 }
191             }
192         }
193 
194         /// <summary>
195         /// Execute shutdown tasks
196         /// </summary>
197         /// <param name="sender">object sending the event</param>
198         /// <param name="e">event arguments</param>
199         private void MainWindow_Closing(object sender, CancelEventArgs e)
200         {
201             if (this.multiReader != null)
202             {
203                 // ColorFrameReder is IDisposable
204                 this.multiReader.Dispose();
205                 this.multiReader = null;
206             }
207 
208             if (this.kinectSensor != null)
209             {
210                 this.kinectSensor.Close();
211                 this.kinectSensor = null;
212             }
213 
214             //initialize a StreamWriter to write the joints positions of all
                    frames into single file
215             string pathCoord = System.IO.Path.Combine(@"C:/Users/Mashaan_Awad/
                    Documents/MATLAB/Data Actual/Raw/Punch_RawCoord.txt");
216             StreamWriter swCoord = new StreamWriter(pathCoord);
217             //search the depth data and add it to the file
218             for (int i = 0; i < this.coordArray.GetLength(0); i++)
219             {
220                 for (int j = 0; j < this.coordArray.GetLength(1); j++)
221                 {
222                     swCoord.Write(coordArray[i, j] + " ");
223                 }
224                 swCoord.Write("\n");
225             }
226             //dispose of sw
227             swCoord.Close();
228 
```

```
229             int MaxDepthValue = int.MaxValue;
230             while (this.depthDataCounter1 < depthData.GetLength(0))
231             {
232                 //initialize a StreamWriter to write the depth values for each
                        frame into a separate file
233                 string pathDepth = System.IO.Path.Combine(@"C:/Users/Mashaan_Awad
                        /Documents/MATLAB/Data Actual/Raw/Punch_RawDepth" +
                        depthDataCounter1 + ".txt");
234                 StreamWriter swDepth = new StreamWriter(pathDepth);
235 
236                 //search the depth data and add it to the file
237                 for (int i = 0; i < this.depthData.GetLength(1); i++)
238                 {
239                     if (this.bodyIndexData[this.depthDataCounter1, i] != 0xff)
240                     {
241                         swDepth.WriteLine(this.depthData[this.depthDataCounter1,
                                i] + "\n"); //\n for a new line
242                     }
243                     else
244                     {
245                         swDepth.WriteLine(MaxDepthValue + "\n"); //\n for a new
                                line
246                     }
247                 }
248 
249                 //dispose of sw
250                 swDepth.Close();
251                 this.depthDataCounter1++;
252             }
253         }
254 
255         /// <summary>
256         /// Handles the color frame data arriving from the sensor
257         /// </summary>
258         /// <param name="sender">object sending the event</param>
259         /// <param name="e">event arguments</param>
260         private void Reader_MultiSourceFrameArrived(object sender,
                MultiSourceFrameArrivedEventArgs e)
261         {
262             bool dataReceived = false;
263             //bool depthFrameProcessed = false;
264             var reference = e.FrameReference.AcquireFrame();
265             // ColorFrame is IDisposable
266 
267             var colorFrame = reference.ColorFrameReference.AcquireFrame();
268             var depthFrame = reference.DepthFrameReference.AcquireFrame();
269             var bodyIndexFrame = reference.BodyIndexFrameReference.AcquireFrame()
                    ;
270             var bodyFrame = reference.BodyFrameReference.AcquireFrame();
271 
272             //using (ColorFrame colorFrame = reference.ColorFrameReference.
                    AcquireFrame())
273             //using (DepthFrame depthFrame = reference.DepthFrameReference.
                    AcquireFrame())
274             //using (BodyIndexFrame bodyIndexFrame = reference.
                    BodyIndexFrameReference.AcquireFrame())
275             //using (BodyFrame bodyFrame = reference.BodyFrameReference.
                    AcquireFrame())
276             //{
277             if (colorFrame != null && depthFrame != null && bodyFrame != null &&
                    bodyIndexFrame != null)
278             {
279                 FrameDescription colorFrameDescription = colorFrame.
                        FrameDescription;
280                 using (KinectBuffer colorBuffer = colorFrame.LockRawImageBuffer()
                        )
281                 {
282                     this.colorBitmap.Lock();
```

```
283                     // verify data and write the new color frame data to the
                            display bitmap
284                     if ((colorFrameDescription.Width == this.colorBitmap.
                            PixelWidth) && (colorFrameDescription.Height == this.
                            colorBitmap.PixelHeight))
285                     {
286                         colorFrame.CopyConvertedFrameDataToIntPtr(
287                             this.colorBitmap.BackBuffer,
288                             (uint)(colorFrameDescription.Width *
                                    colorFrameDescription.Height * 4),
289                             ColorImageFormat.Bgra);
290                         this.colorBitmap.AddDirtyRect(new Int32Rect(0, 0, this.
                                colorBitmap.PixelWidth, this.colorBitmap.PixelHeight)
                                );
291                     }
292                     this.colorBitmap.Unlock();
293                 }
294
295                 if (bodyFrame != null)
296                 {
297
298                     if (this.bodies == null)
299                     {
300                         this.bodies = new Body[bodyFrame.BodyCount];
301                     }
302
303                     // The first time GetAndRefreshBodyData is called, Kinect
                            will allocate each Body in the array.
304                     // As long as those body objects are not disposed and not set
                            to null in the array,
305                     // those body objects will be re-used.
306                     bodyFrame.GetAndRefreshBodyData(this.bodies);
307                     dataReceived = true;
308                 }
309
310                 if (dataReceived == true && frameCounter % 2 == 0 &&
                        recordCounter < this.depthData.GetLength(0))
311                 {
312                     Body body = this.bodies[0];
313                     Console.WriteLine("frameCounter = " + frameCounter);
314                     //if (frameCounter % 5 == 0 && recordCounter < this.depthData
                            .GetLength(0))
315                     if (body.IsTracked)
316                     {
317                         Console.WriteLine("copy body joints data");
318                         IReadOnlyDictionary<JointType, Joint> joints = body.
                                Joints;
319                         foreach (JointType jointType in joints.Keys)
320                         {
321                             // sometimes the depth(Z) of an inferred joint may
                                    show as negative
322                             // clamp down to 0.1f to prevent coordinatemapper
                                    from returning (-Infinity, -Infinity)
323                             CameraSpacePoint position = joints[jointType].
                                    Position;
324                             if (position.Z < 0)
325                             {
326                                 position.Z = InferredZPositionClamp;
327                             }
328                             //if (frameCounter % 10 == 0 && coordDataCounter1 <
                                    this.coordArray.GetLength(0))
329                             //{
330                             //coordRecorded = true;
331                             //Console.WriteLine("copy positions data");
332                             int coordPos = 0;
333                             switch (jointType)
334                             {
335                                 case JointType.Head:
```

```
336                     coordPos = 0;
337                     coordArray[recordCounter, coordPos] =
                            position.X;
338                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
339                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
340                     break;
341                 case JointType.Neck:
342                     coordPos = 3;
343                     coordArray[recordCounter, coordPos] =
                            position.X;
344                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
345                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
346                     break;
347                 case JointType.SpineMid:
348                     coordPos = 6;
349                     coordArray[recordCounter, coordPos] =
                            position.X;
350                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
351                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
352                     break;
353                 case JointType.ShoulderLeft:
354                     coordPos = 9;
355                     coordArray[recordCounter, coordPos] =
                            position.X;
356                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
357                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
358                     break;
359                 case JointType.ElbowLeft:
360                     coordPos = 12;
361                     coordArray[recordCounter, coordPos] =
                            position.X;
362                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
363                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
364                     break;
365                 case JointType.HandLeft:
366                     coordPos = 15;
367                     coordArray[recordCounter, coordPos] =
                            position.X;
368                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
369                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
370                     break;
371                 case JointType.ShoulderRight:
372                     coordPos = 18;
373                     coordArray[recordCounter, coordPos] =
                            position.X;
374                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
375                     coordArray[recordCounter, coordPos + 2] =
                            position.Z;
376                     break;
377                 case JointType.ElbowRight:
378                     coordPos = 21;
379                     coordArray[recordCounter, coordPos] =
                            position.X;
380                     coordArray[recordCounter, coordPos + 1] =
                            position.Y;
```

```
381                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
382                         break;
383                     case JointType.HandRight:
384                         coordPos = 24;
385                         coordArray[recordCounter, coordPos] =
                                position.X;
386                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
387                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
388                         break;
389                     case JointType.HipLeft:
390                         coordPos = 27;
391                         coordArray[recordCounter, coordPos] =
                                position.X;
392                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
393                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
394                         break;
395                     case JointType.KneeLeft:
396                         coordPos = 30;
397                         coordArray[recordCounter, coordPos] =
                                position.X;
398                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
399                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
400                         break;
401                     case JointType.FootLeft:
402                         coordPos = 33;
403                         coordArray[recordCounter, coordPos] =
                                position.X;
404                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
405                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
406                         break;
407                     case JointType.HipRight:
408                         coordPos = 36;
409                         coordArray[recordCounter, coordPos] =
                                position.X;
410                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
411                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
412                         break;
413                     case JointType.KneeRight:
414                         coordPos = 39;
415                         coordArray[recordCounter, coordPos] =
                                position.X;
416                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
417                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
418                         break;
419                     case JointType.FootRight:
420                         coordPos = 42;
421                         coordArray[recordCounter, coordPos] =
                                position.X;
422                         coordArray[recordCounter, coordPos + 1] =
                                position.Y;
423                         coordArray[recordCounter, coordPos + 2] =
                                position.Z;
424                         break;
425                     default:
426                         break;
```

```
427                                 }
428                                 //}
429
430                                 //DepthSpacePoint depthSpacePoint = this.
                                        coordinateMapper.MapCameraPointToDepthSpace(
                                        position);
431                             }
432
433                             Console.WriteLine("copy depth data");
434                             depthFrame.CopyFrameDataToArray(tempDepthData);
435                             //search the depth data and add it to the file
436                             for (int i = 0; i < this.tempDepthData.Length; i++)
437                             {
438                                 this.depthData[recordCounter, i] = this.tempDepthData
                                        [i];
439                             }
440
441                             Console.WriteLine("copy body index data");
442                             bodyIndexFrame.CopyFrameDataToArray(tempBodyIndexData);
443                             //search the depth data and add it to the file
444                             for (int i = 0; i < this.tempBodyIndexData.Length; i++)
445                             {
446                                 this.bodyIndexData[recordCounter, i] = this.
                                        tempBodyIndexData[i];
447                             }
448                             recordCounter++;
449                         }
450                     }
451                     frameCounter++;
452                     colorFrame.Dispose();
453                     depthFrame.Dispose();
454                     bodyIndexFrame.Dispose();
455                     bodyFrame.Dispose();
456                 }
457                 //}
458             }
459
460
461             /// <summary>
462             /// Handles the event which the sensor becomes unavailable (E.g. paused,
                    closed, unplugged).
463             /// </summary>
464             /// <param name="sender">object sending the event</param>
465             /// <param name="e">event arguments</param>
466             private void Sensor_IsAvailableChanged(object sender,
                    IsAvailableChangedEventArgs e)
467             {
468                 // on failure, set the status text
469                 this.StatusText = this.kinectSensor.IsAvailable
470                 ? Properties.Resources.RunningStatusText
471                 : Properties.Resources.SensorNotAvailableStatusText;
472             }
473         }
474     }
```

A.2 Draw Depth Images using Recorded Depth Data

The depth data generated by the previous section are raw numbers that indicate the distance to the surface reserved by a particular pixel. Therefore, visualizing these depth numbers is mandatory to perform some tasks (e.g. segmentation).

Figure A.1: Grayscale Depth Image with Subtracted Background

The script shown in Listing A.2 reads the text files that contain depth information and generate the corresponding grayscale images (see Figure A.1).

Listing A.2: Matlab script to Draw Depth Images from Depth Data Recorded by Kinect

```
% this script reads text files generated by kinect
% then draws a grayscale images out of them


files = dir('*.txt');
for FileCounter=1:length(files)
    x = load(files(FileCounter).name);
    y=reshape(x,512,424)
    for i=1:size(y,1)
        for j=1:size(y,2)
            if y(i,j)<500 && y(i,j)>4500
                y(i,j) = 0;
            else
                y(i,j) = round(y(i,j)/(8000 / 256));
            end
        end
    end
%     imshow(y,[0 255]);
    I = mat2gray(y, [0 255]);
    I = imrotate(I,270);
%     imwrite(I , [num2str(FileCounter) '.png'])
    imwrite(I , [strrep(files(FileCounter).name, '.txt', '') '.png'])
end
```

A.3 Segment and Preprocess Joints Positions Based on Depth Images Observations

After generating the grayscale depth images, we go over them to determine the start and end frames of each action. Afterwards, we segmented the world positions into text files where each file contain a set of frames that represent a particular

action. Moreover, we preprocess the segmented actions, the preprocessing contains:

- **Translate:** find the centroid of all coordinates then subtract it from all coordinates to move the player into the middle of the scene (see line 31 to 65 of Listing A.3).
- **Normalize:** all the coordinates divided by the distance between the neck and the spinemid (see line 67 to 71 of Listing A.3).
- **Rotate:** this step was performed only on the real data recorded by the Kinect because the Kinect behaves as a mirror to provide a convenient usage. Therefore, we have to rotate the player over the Y axis (see line 73 to 79 of Listing A.3).

Listing A.3: Matlab script to Segment and Preprocess Joints Positions Recorded by Kinect

```
1  % this script reads raw coordinates populated by kinect
2  % and segment them into samples based on index array
3  % the index array is filled after looking at depth images
4  % also this script preprocess the samples by performing translation,
5  % normalization, and rotation
6
7  SubjectID = 1;
8  action = 'Stretch';
9  RawCoord = load([action '_RawCoord.txt']);
10 NumOfJoints = 15;
11 NumOfDimension = 3;
12 NumOfFeatures = NumOfJoints*NumOfDimension;
13 StartIndexCoord = [65 195 320 425 540 655 760 865
       975 1085];
14 StartIndexCoord = StartIndexCoord+1;
15 EndIndexCoord = [140 265 380 495 595 715 820 935
       1040 1145];
16 EndIndexCoord = EndIndexCoord+1;
17 NameCounter = 1;
18
19 for IndexCounter=1:length(StartIndexCoord)
20     CurrentInstance = RawCoord(StartIndexCoord(IndexCounter):EndIndexCoord(
           IndexCounter),:);
21
22     NumOfFrames = size(CurrentInstance,1);
23     x = zeros(NumOfFrames,(NumOfFeatures/3));
24     y = zeros(NumOfFrames,(NumOfFeatures/3));
25     z = zeros(NumOfFrames,(NumOfFeatures/3));
26     xMean = zeros(NumOfFrames,1);
27     yMean = zeros(NumOfFrames,1);
28     zMean = zeros(NumOfFrames,1);
29     coordinates1 = zeros(NumOfFrames,NumOfFeatures);
30
31     colCounter=1;
32     for k=1:3:43
33         x(:,colCounter)=CurrentInstance(:,k);
34         colCounter=colCounter+1;
35     end
36     colCounter=1;
37     for k=2:3:44
38         y(:,colCounter)=CurrentInstance(:,k);
39         colCounter=colCounter+1;
40     end
```

```
41      colCounter=1;
42      for k=3:3:45
43          z(:,colCounter)=CurrentInstance(:,k);
44          colCounter=colCounter+1;
45      end
46      for k=1:NumOfFrames
47          xMean(k,1)=(sum(x(k,:))/15);
48      end
49      for k=1:NumOfFrames
50          yMean(k,1)=(sum(y(k,:))/15);
51      end
52      for k=1:NumOfFrames
53          zMean(k,1)=(sum(z(k,:))/15);
54      end
55
56      coordinates1 = CurrentInstance;
57      for k=1:3:43
58          coordinates1(:,k) = coordinates1(:,k)-xMean;
59      end
60      for k=2:3:44
61          coordinates1(:,k) = coordinates1(:,k)-yMean;
62      end
63      for k=3:3:45
64          coordinates1(:,k) = coordinates1(:,k)-zMean;
65      end
66
67      xDiff = (CurrentInstance(1,4)-CurrentInstance(1,7))^2;
68      yDiff = (CurrentInstance(1,5)-CurrentInstance(1,8))^2;
69      zDiff = (CurrentInstance(1,6)-CurrentInstance(1,9))^2;
70      dist = sqrt(xDiff+yDiff+zDiff);
71      coordinates1 = coordinates1/dist;
72
73      for RotateCounter=1:3:size(coordinates1,2)
74          angleY = 110;
75          Ry = [cos(angleY) 0 sin(angleY); 0 1 0; -sin(angleY) 0 cos(angleY)]; %
                rotation matrix for y-axis
76          RotateCoord = coordinates1(:,RotateCounter:RotateCounter+2)';
77          RotateCoord = Ry*RotateCoord;
78          coordinates1(:,RotateCounter:RotateCounter+2) = RotateCoord';
79      end
80
81      dlmwrite(['Subject' num2str(SubjectID) '_' action '_PreprocessSample' num2str
            (NameCounter) '.txt'],coordinates1,'delimiter','\t')
82      NameCounter = NameCounter+1;
83  end
```

APPENDIX B

GENERATE AND PREPROCESS SYNTHETIC HUMAN ACTION DATA

B.1 Create Human Characters

Creating a human character from scratch is an intensive task that required huge effort. Fortunately, there are tools to generate basic human characters such as AutoDesk Character Generator which we used in this tutorial. Go to the link below then sign in with your AutoDesk ID.:

`https://charactergenerator.autodesk.com/`

Click new at the top left corner as shown in Figure B.1.

Figure B.2 shows the basic models provided by the tool, choose your basic model then click customize.

Now, you can edit your character, a pre-made designs appears in the right handside you can use them or you can customize them by changing the values. After editing click finish (see Figure B.3).

Name your character as shown in Figure B.4.

The character will appear in your home screen and you need to export it by clicking the arrow as illustrated in Figure B.5.

Specify the exporting options such as the resolution and character height. Note that some options (e.g. face expressions) need a premium account to be selected. Also, it's recommended to export the character as .fbx file (see Figure B.6).

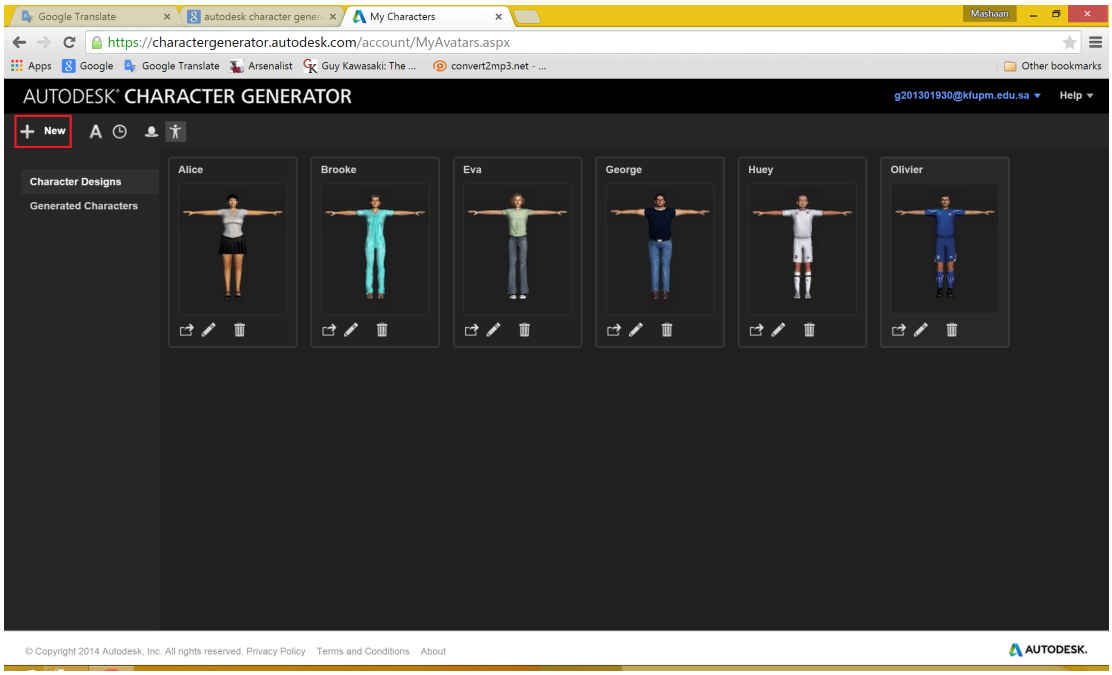


Figure B.1: New Character in AutoDesk Character Generator

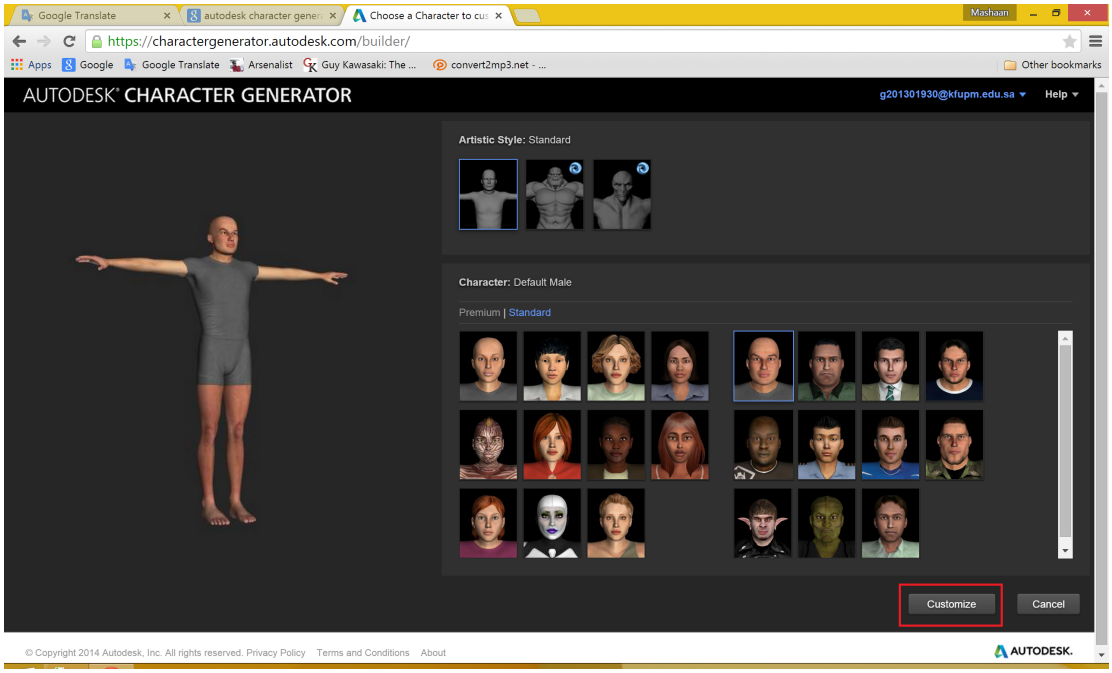


Figure B.2: Basic Models in AutoDesk Character Generator

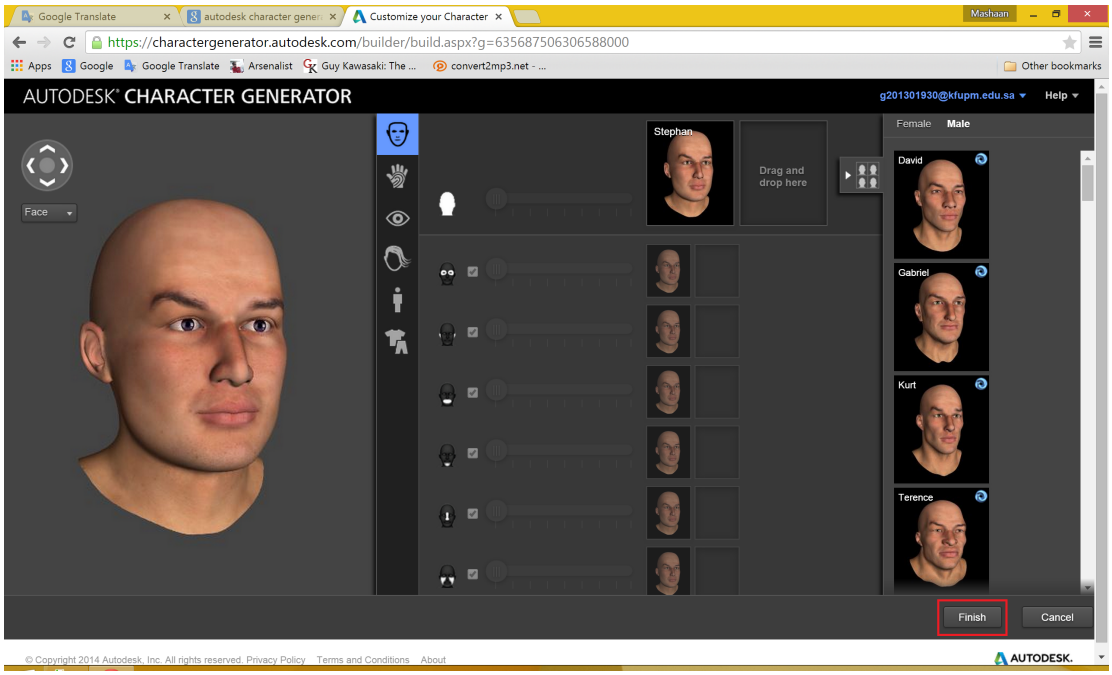


Figure B.3: Editing a Model in AutoDesk Character Generator

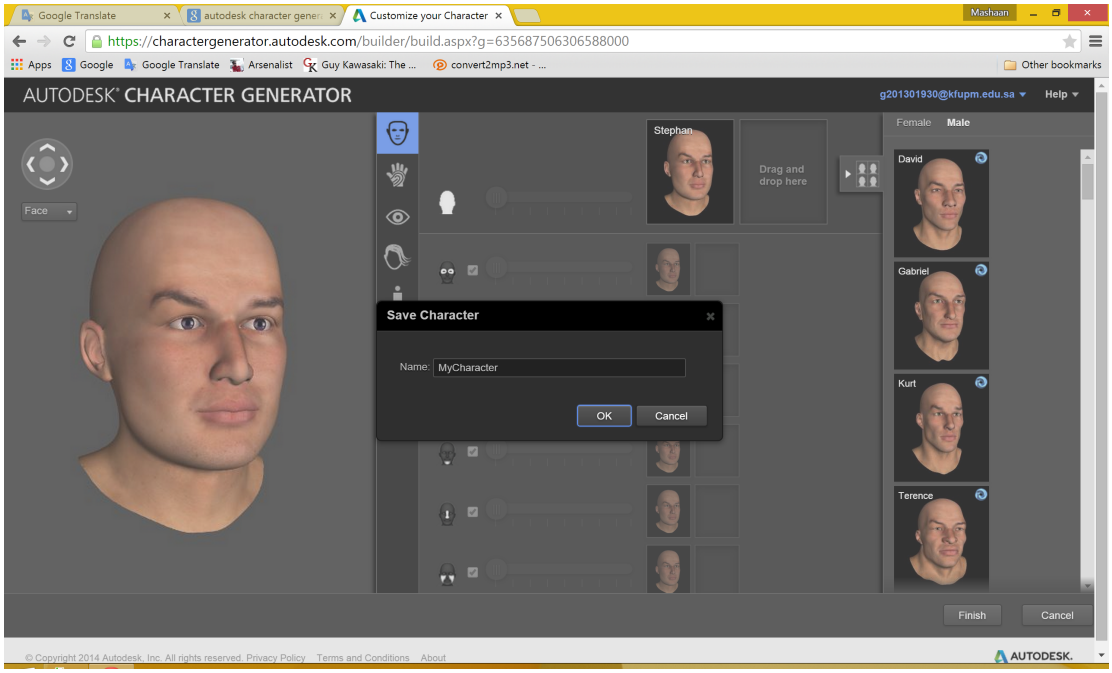


Figure B.4: Finalizing a Model in AutoDesk Character Generator

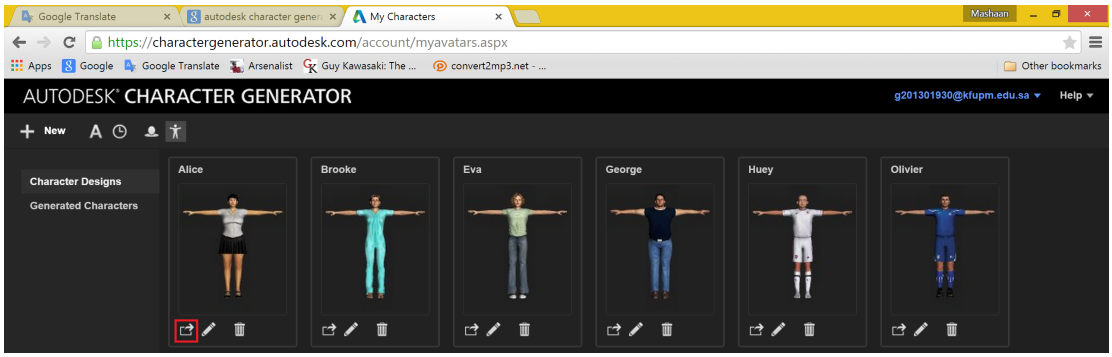


Figure B.5: Customized Models in AutoDesk Character Generator

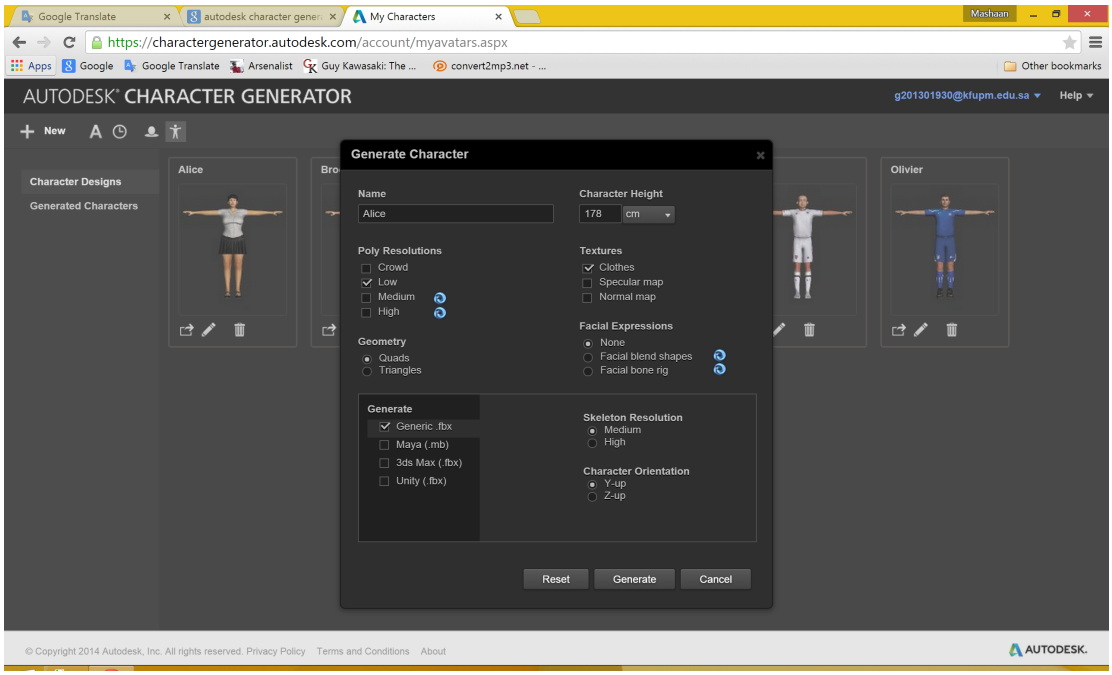


Figure B.6: Exporting Models in AutoDesk Character Generator

B.2 Apply Maya HumanIK to a Human Character

The human character generated in the previous subsection is rigged (i.e., the mesh is skinned to a joints for deformation). However, this skeleton should be processed by Maya HumanIK to provide a natural human deformation after applying the Motion Capture (MoCap) files. Open a new Maya scene the choose New>Import to import the character generated in the previous subsection. As illustrated in Figure B.7, open the outliner and expand the group that contains your character components.

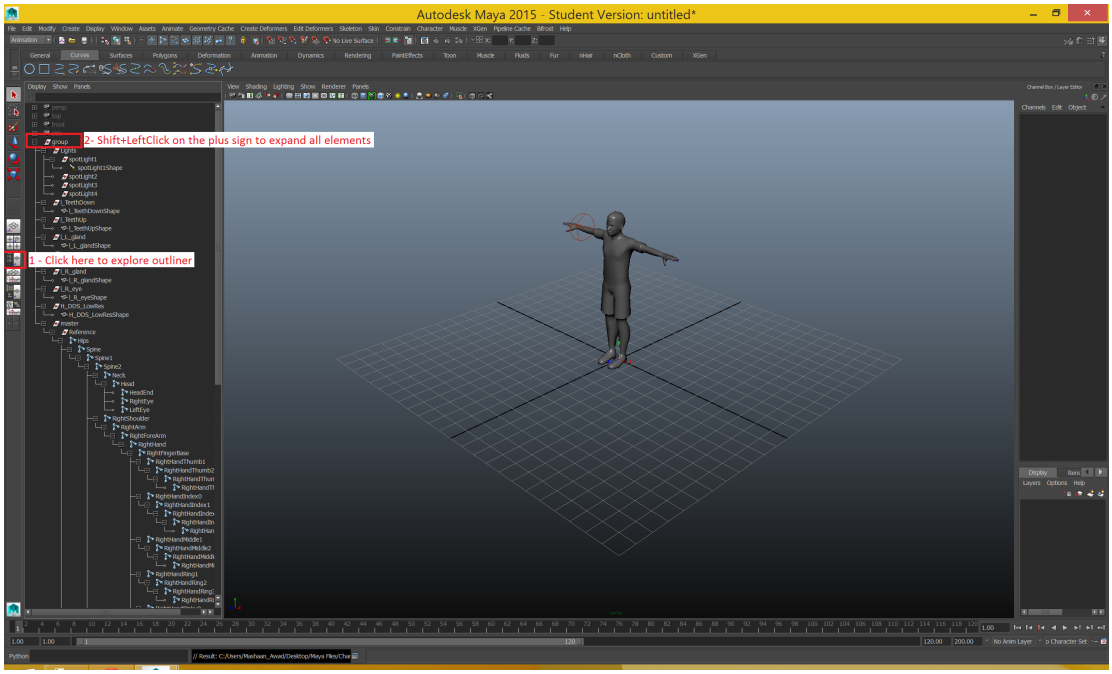


Figure B.7: Importing the generated model to Maya

Select Skeleton>HumanIK, from the HumanIK window then select Define>Skeleton, an unassigned skeleton will show up. Our task is to assign the bones shown in the HumanIK skeleton to their corresponding bones in our imported character. Apply the steps illustrated in Figure B.8 to assign bones to HumanIK, enabling mirror mode can be helpful to save time. Mirror mode automatically assigns the bone on the opposite side (e.g. left arm and right arm).

Once you assign all bones successfully the HumanIK skeleton turns to a green color indicating a successful assignment (as shown in Figure B.9). To save time and effort of assigning the character each time you import it, save the assigned character as maya binary file (.mb) so you can import a HumanIK ready version of your character. Go to File>Save as, to save your character as .mb file.

B.3 Apply Maya HumanIK to a Motion Capture File

Motion Capture files (or simply MoCaps) are the files that contain motion information for models in computer graphics software. There are numerous extensions of MoCaps, however the extension we implemented in this tutorial is (.fbx). One can download free MoCaps from NUS (National University of Singapore) Motion Capture Database by visiting the link below:

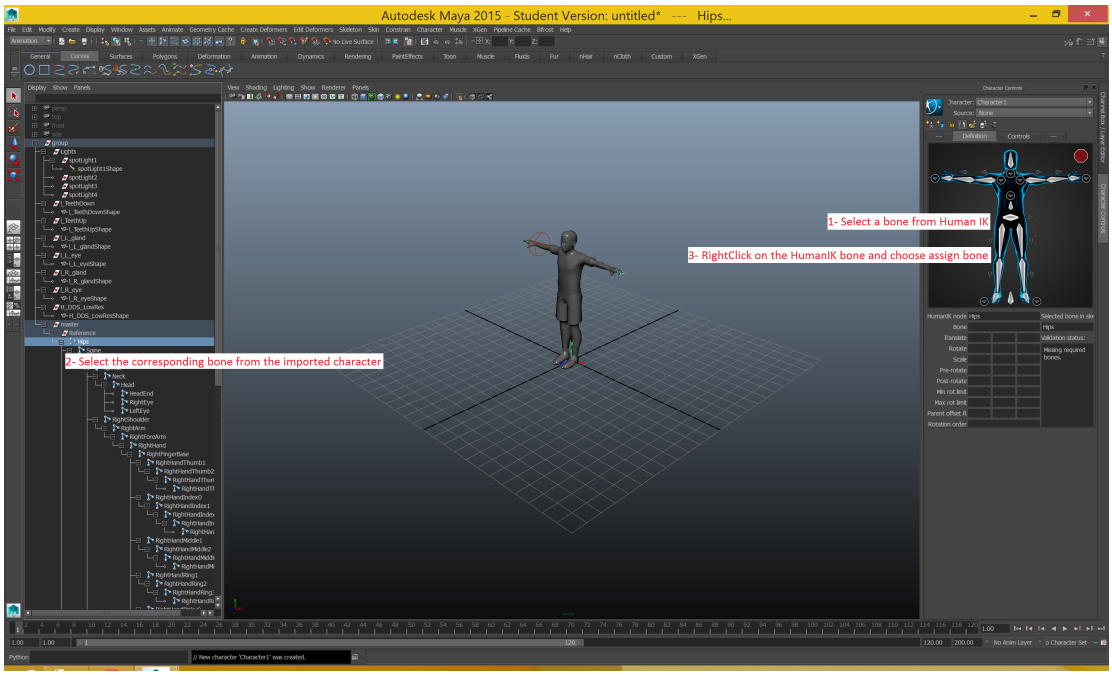


Figure B.8: Assign the imported skeleton to a HumanIK skeleton

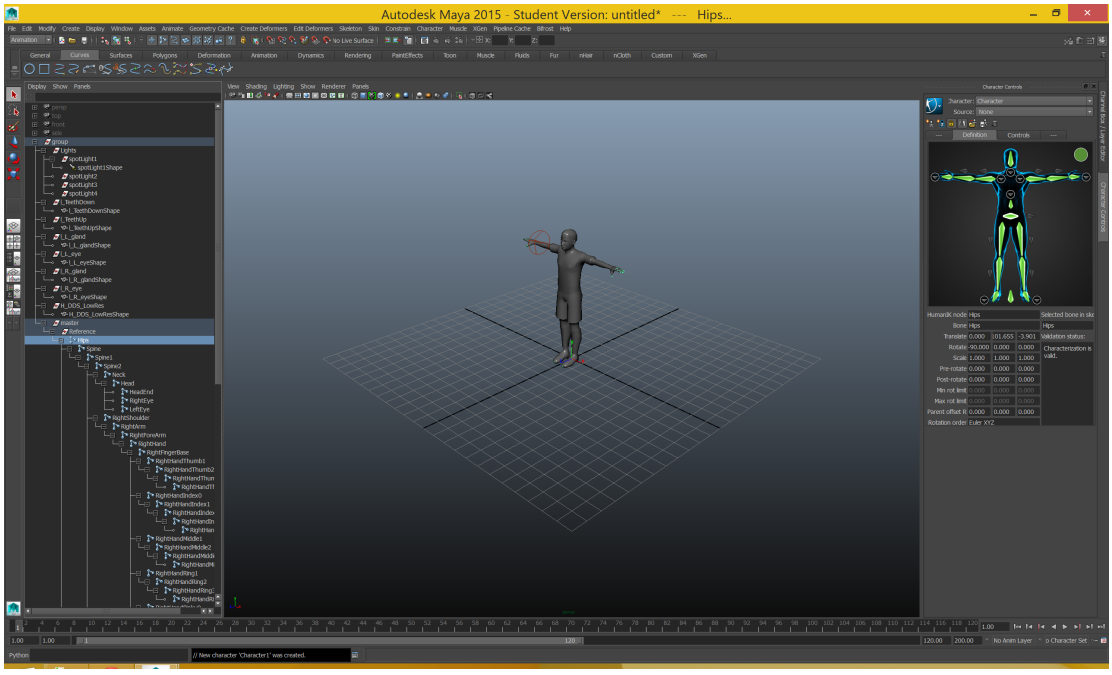


Figure B.9: Successful Assignment of a HumanIK skeleton

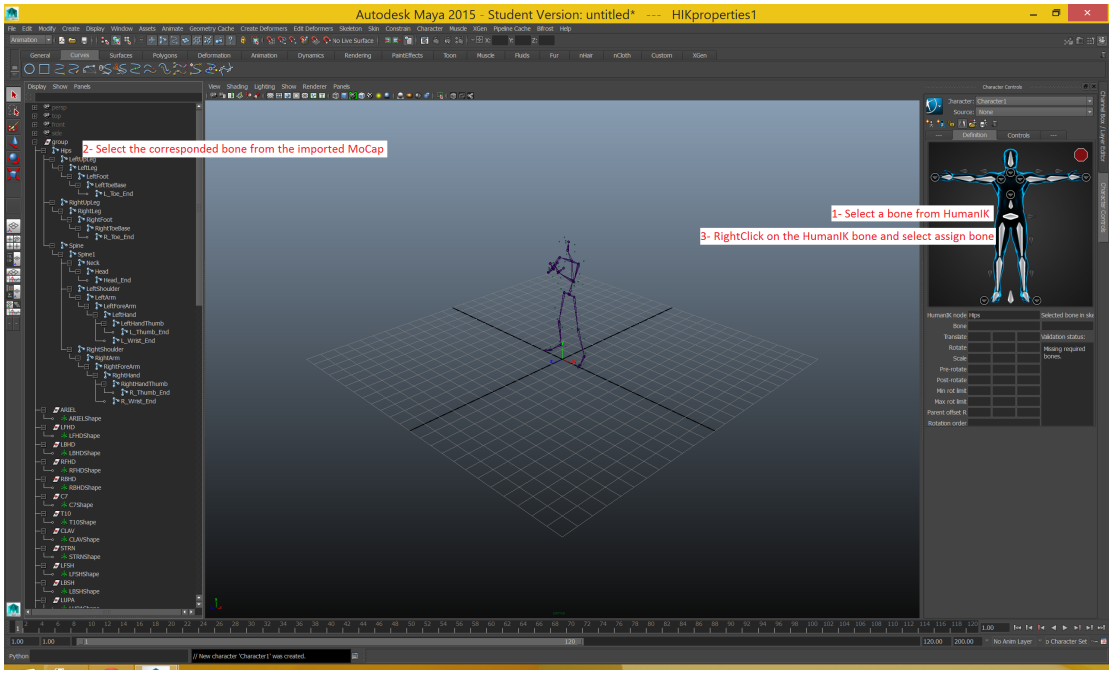


Figure B.10: Assign the imported MoCap to a HumanIK skeleton

`http://animation.comp.nus.edu.sg/nusmocap.html`

The MoCaps also should be assigned to Maya HumanIK to provide a natural deformation for the human characters. Apply the steps shown in Figure B.10 to assign MoCap bones to HumanIK bones.

A successful assignment converts the HumanIK skeleton into green as shown in Figure B.11. To save time and effort of assigning the MoCap each time you import it, save the assigned MoCap as maya binary file (.mb) so you can import a HumanIK ready version of your MoCap. Go to File>Save as, to save your character as .mb file.

B.4 Apply Motion Capture File to a Human Character

The outcomes of the previous two subsection are:

- .mb file contains a human character assigned to a HumanIK skeleton.

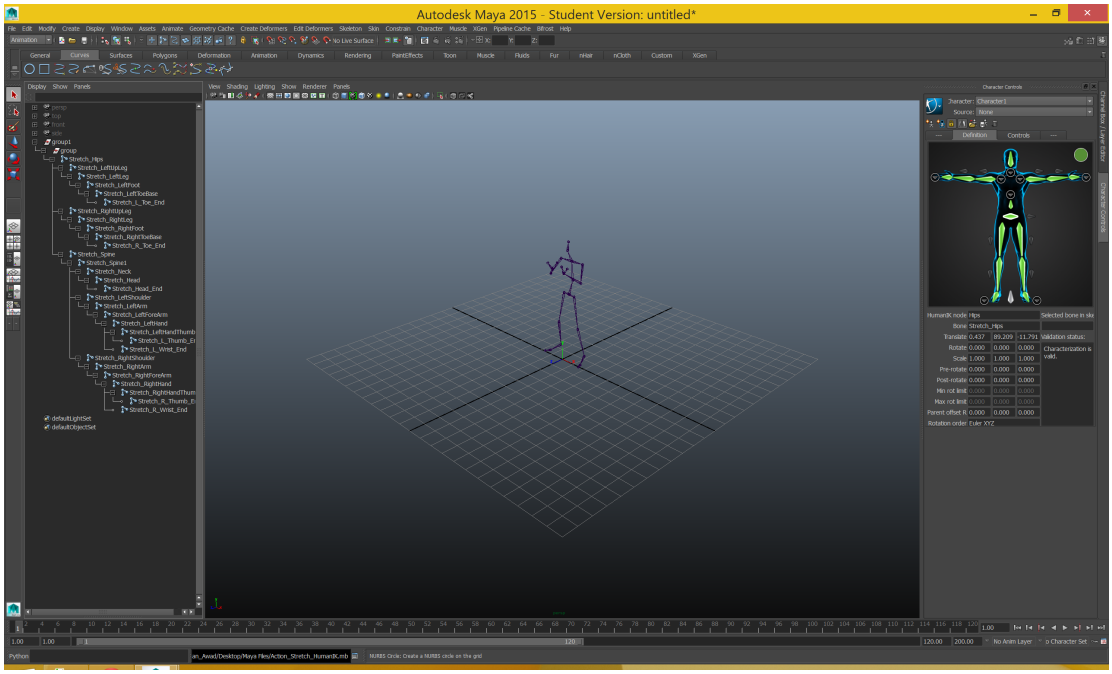


Figure B.11: Successful Assignment of a HumanIK skeleton

- .mb file contains a MoCap assigned to a HumanIK skeleton.

In this subsection, we aim to apply the motion of the MoCap to the human model such that it deforms naturally. Now, you need to import aforementioned .mb files to an empty Maya scene.

WARNING: The joints names of the human character must be different than the ones exist in MoCap, otherwise it might produce undesirable results. In most cases, Maya takes care of that using namespace property but this is not always the case.

After importing both .mb files, open the HumanIK window and choose your human model from the character drop down menu. Then, choose the MoCap file from the source drop down menu.

B.5 Install Multiple Cameras in a Maya Scene

To capture a human action from different perspectives in a Maya scene, we need to use camera objects. Nevertheless, creating multiple cameras manually can be an intensive task. Therefore, we provide a python script (see Listing B.1) that installs 27 cameras according to the spherical coordinates around the human character spine. The cameras positions are provided in Table B.1. The generated cameras should appear as shown in Figure B.13.

Listing B.1: Python script to generate 27 cameras with respect to a human spine

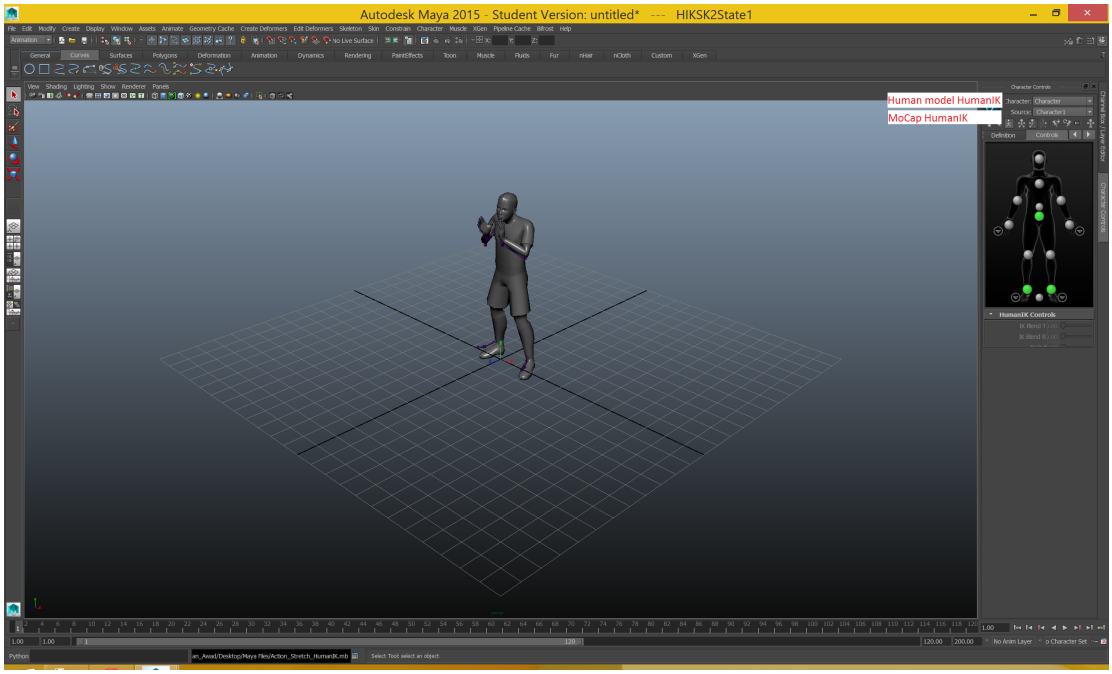


Figure B.12: Apply MoCap to a Human Model

Table B.1: Cameras Positions

	r	θ	φ		r	θ	φ		r	θ	φ
Camera 1	300	0°	0°	**Camera 10**	300	90°	30°	**Camera 19**	300	90°	-30°
Camera 2	300	0°	10°	**Camera 11**	300	70°	30°	**Camera 20**	300	70°	-30°
Camera 3	300	0°	20°	**Camera 12**	300	50°	30°	**Camera 21**	300	50°	-30°
Camera 4	300	0°	30°	**Camera 13**	300	30°	30°	**Camera 22**	300	30°	-30°
Camera 5	300	0°	40°	**Camera 14**	300	10°	30°	**Camera 23**	300	10°	-30°
Camera 6	300	0°	-10°	**Camera 15**	300	110°	30°	**Camera 24**	300	110°	-30°
Camera 7	300	0°	-20°	**Camera 16**	300	130°	30°	**Camera 25**	300	130°	-30°
Camera 8	300	0°	-30°	**Camera 17**	300	150°	30°	**Camera 26**	300	150°	-30°
Camera 9	300	0°	-40°	**Camera 18**	300	170°	30°	**Camera 27**	300	170°	-30°

```
1  import math
2  import maya.mel
3  import maya.cmds as cmds
4  # save the spine coordinates of the human model
5  SpineCoord = maya.mel.eval("xform -q -t -ws Spine1 ;");
6  r = SpineCoord[2]+300;
7  Theta = math.radians(0);
8
9  Matrix1 = [[0 for x in range(3)] for x in range(9)];
10
11 Matrix1[0][0]=r*math.cos(Theta)*math.sin(math.radians(0)); Matrix1[0][1]=r*math.
       sin(Theta)*math.sin(math.radians(0)); Matrix1[0][2]=r*math.cos(math.radians
       (0));
12 Matrix1[1][0]=r*math.cos(Theta)*math.sin(math.radians(10)); Matrix1[1][1]=r*math.
       sin(Theta)*math.sin(math.radians(10)); Matrix1[1][2]=r*math.cos(math.radians
       (10));
13 Matrix1[2][0]=r*math.cos(Theta)*math.sin(math.radians(20)); Matrix1[2][1]=r*math.
       sin(Theta)*math.sin(math.radians(20)); Matrix1[2][2]=r*math.cos(math.radians
       (20));
14 Matrix1[3][0]=r*math.cos(Theta)*math.sin(math.radians(30)); Matrix1[3][1]=r*math.
```

Figure B.13: Setup of generated cameras

```
    sin(Theta)*math.sin(math.radians(30)); Matrix1[3][2]=r*math.cos(math.radians
    (30));
15  Matrix1[4][0]=r*math.cos(Theta)*math.sin(math.radians(40)); Matrix1[4][1]=r*math.
    sin(Theta)*math.sin(math.radians(40)); Matrix1[4][2]=r*math.cos(math.radians
    (40));
16  Matrix1[5][0]=r*math.cos(Theta)*math.sin(math.radians(-10)); Matrix1[5][1]=r*math
    .sin(Theta)*math.sin(math.radians(-10)); Matrix1[5][2]=r*math.cos(math.
    radians(-10));
17  Matrix1[6][0]=r*math.cos(Theta)*math.sin(math.radians(-20)); Matrix1[6][1]=r*math
    .sin(Theta)*math.sin(math.radians(-20)); Matrix1[6][2]=r*math.cos(math.
    radians(-20));
18  Matrix1[7][0]=r*math.cos(Theta)*math.sin(math.radians(-30)); Matrix1[7][1]=r*math
    .sin(Theta)*math.sin(math.radians(-30)); Matrix1[7][2]=r*math.cos(math.
    radians(-30));
19  Matrix1[8][0]=r*math.cos(Theta)*math.sin(math.radians(-40)); Matrix1[8][1]=r*math
    .sin(Theta)*math.sin(math.radians(-40)); Matrix1[8][2]=r*math.cos(math.
    radians(-40));
20
21
22
23  j = 0;
24  for i in range(1, 10):
25          cmds.camera();
26          maya.mel.eval('setAttr "camera'+str(i)+'.scale" -type "double3" 20 20 20;
                ');
27          maya.mel.eval('setAttr "camera'+str(i)+'.translate" -type "double3"'+str(
                Matrix1[j][0]+SpineCoord[0])+' '+str(Matrix1[j][1]+SpineCoord[1])+' '
                +str(Matrix1[j][2]+SpineCoord[2])+';');
28          maya.mel.eval('setAttr "cameraShape'+str(i)+'.renderable" 1;');
29          maya.mel.eval('setAttr "cameraShape'+str(i)+'.depth" 1;');
30          maya.mel.eval('viewPlace -la '+str(SpineCoord[0])+' '+str(SpineCoord[1])+
                ' '+str(SpineCoord[2])+ '"cameraShape'+str(i)+'";');
31          j += 1;
32
33  Phi = 30;
34
35  Matrix1 = [[0 for x in range(3)] for x in range(9)];
36
```

```
37 Matrix1[0][0]=r*math.cos(math.radians(90))*math.sin(math.radians(Phi)); Matrix1
       [0][1]=r*math.sin(math.radians(90))*math.sin(math.radians(Phi)); Matrix1
       [0][2]=r*math.cos(math.radians(Phi));
38 Matrix1[1][0]=r*math.cos(math.radians(70))*math.sin(math.radians(Phi)); Matrix1
       [1][1]=r*math.sin(math.radians(70))*math.sin(math.radians(Phi)); Matrix1
       [1][2]=r*math.cos(math.radians(Phi));
39 Matrix1[2][0]=r*math.cos(math.radians(50))*math.sin(math.radians(Phi)); Matrix1
       [2][1]=r*math.sin(math.radians(50))*math.sin(math.radians(Phi)); Matrix1
       [2][2]=r*math.cos(math.radians(Phi));
40 Matrix1[3][0]=r*math.cos(math.radians(30))*math.sin(math.radians(Phi)); Matrix1
       [3][1]=r*math.sin(math.radians(30))*math.sin(math.radians(Phi)); Matrix1
       [3][2]=r*math.cos(math.radians(Phi));
41 Matrix1[4][0]=r*math.cos(math.radians(10))*math.sin(math.radians(Phi)); Matrix1
       [4][1]=r*math.sin(math.radians(10))*math.sin(math.radians(Phi)); Matrix1
       [4][2]=r*math.cos(math.radians(Phi));
42 Matrix1[5][0]=r*math.cos(math.radians(110))*math.sin(math.radians(Phi)); Matrix1
       [5][1]=r*math.sin(math.radians(110))*math.sin(math.radians(Phi)); Matrix1
       [5][2]=r*math.cos(math.radians(Phi));
43 Matrix1[6][0]=r*math.cos(math.radians(130))*math.sin(math.radians(Phi)); Matrix1
       [6][1]=r*math.sin(math.radians(130))*math.sin(math.radians(Phi)); Matrix1
       [6][2]=r*math.cos(math.radians(Phi));
44 Matrix1[7][0]=r*math.cos(math.radians(150))*math.sin(math.radians(Phi)); Matrix1
       [7][1]=r*math.sin(math.radians(150))*math.sin(math.radians(Phi)); Matrix1
       [7][2]=r*math.cos(math.radians(Phi));
45 Matrix1[8][0]=r*math.cos(math.radians(170))*math.sin(math.radians(Phi)); Matrix1
       [8][1]=r*math.sin(math.radians(170))*math.sin(math.radians(Phi)); Matrix1
       [8][2]=r*math.cos(math.radians(Phi));
46
47
48
49 j = 0;
50 for i in range(10, 19):
51         cmds.camera();
52         maya.mel.eval('setAttr "camera'+str(i)+'.scale" -type "double3" 20 20 20;
               ');
53         maya.mel.eval('setAttr "camera'+str(i)+'.translate" -type "double3"'+str(
               Matrix1[j][0]+SpineCoord[0])+' '+str(Matrix1[j][1]+SpineCoord[1])+' '
               +str(Matrix1[j][2]+SpineCoord[2])+';');
54         maya.mel.eval('setAttr "cameraShape'+str(i)+'.renderable" 1;');
55         maya.mel.eval('setAttr "cameraShape'+str(i)+'.depth" 1;');
56         maya.mel.eval('viewPlace -la '+str(SpineCoord[0])+' '+str(SpineCoord[1])+
               ' '+str(SpineCoord[2])+ '"cameraShape'+str(i)+'";');
57         j += 1;
58
59
60
61 Phi = -30;
62
63 Matrix1 = [[0 for x in range(3)] for x in range(9)];
64
65 Matrix1[0][0]=r*math.cos(math.radians(90))*math.sin(math.radians(Phi)); Matrix1
       [0][1]=r*math.sin(math.radians(90))*math.sin(math.radians(Phi)); Matrix1
       [0][2]=r*math.cos(math.radians(Phi));
66 Matrix1[1][0]=r*math.cos(math.radians(70))*math.sin(math.radians(Phi)); Matrix1
       [1][1]=r*math.sin(math.radians(70))*math.sin(math.radians(Phi)); Matrix1
       [1][2]=r*math.cos(math.radians(Phi));
67 Matrix1[2][0]=r*math.cos(math.radians(50))*math.sin(math.radians(Phi)); Matrix1
       [2][1]=r*math.sin(math.radians(50))*math.sin(math.radians(Phi)); Matrix1
       [2][2]=r*math.cos(math.radians(Phi));
68 Matrix1[3][0]=r*math.cos(math.radians(30))*math.sin(math.radians(Phi)); Matrix1
       [3][1]=r*math.sin(math.radians(30))*math.sin(math.radians(Phi)); Matrix1
       [3][2]=r*math.cos(math.radians(Phi));
69 Matrix1[4][0]=r*math.cos(math.radians(10))*math.sin(math.radians(Phi)); Matrix1
       [4][1]=r*math.sin(math.radians(10))*math.sin(math.radians(Phi)); Matrix1
       [4][2]=r*math.cos(math.radians(Phi));
70 Matrix1[5][0]=r*math.cos(math.radians(110))*math.sin(math.radians(Phi)); Matrix1
       [5][1]=r*math.sin(math.radians(110))*math.sin(math.radians(Phi)); Matrix1
       [5][2]=r*math.cos(math.radians(Phi));
```

```
71 Matrix1[6][0]=r*math.cos(math.radians(130))*math.sin(math.radians(Phi)); Matrix1
       [6][1]=r*math.sin(math.radians(130))*math.sin(math.radians(Phi)); Matrix1
       [6][2]=r*math.cos(math.radians(Phi));
72 Matrix1[7][0]=r*math.cos(math.radians(150))*math.sin(math.radians(Phi)); Matrix1
       [7][1]=r*math.sin(math.radians(150))*math.sin(math.radians(Phi)); Matrix1
       [7][2]=r*math.cos(math.radians(Phi));
73 Matrix1[8][0]=r*math.cos(math.radians(170))*math.sin(math.radians(Phi)); Matrix1
       [8][1]=r*math.sin(math.radians(170))*math.sin(math.radians(Phi)); Matrix1
       [8][2]=r*math.cos(math.radians(Phi));
74
75
76
77 j = 0;
78 for i in range(19, 28):
79         cmds.camera();
80         maya.mel.eval('setAttr "camera'+str(i)+'.scale" -type "double3" 20 20 20;
               ');
81         maya.mel.eval('setAttr "camera'+str(i)+'.translate" -type "double3" '+str(
               Matrix1[j][0]+SpineCoord[0])+' '+str(Matrix1[j][1]+SpineCoord[1])+' '
               +str(Matrix1[j][2]+SpineCoord[2])+';');
82         maya.mel.eval('setAttr "cameraShape'+str(i)+'.renderable" 1;');
83         maya.mel.eval('setAttr "cameraShape'+str(i)+'.depth" 1;');
84         maya.mel.eval('viewPlace -la '+str(SpineCoord[0])+' '+str(SpineCoord[1])+
               ' '+str(SpineCoord[2])+ '"cameraShape'+str(i)+'";');
85         j += 1;
```

B.6 Export Human Joints Positions During Action

After cameras installations we need to capture the joints positions during the movement of the human character. The python script provided in Listing B.2 generates two text files: one for the human joints world positions and the other one for the cameras world positions. Then we use Matlab to generate the samples related to each cameras. The python script (see Listing B.2) pulls out the positions of 15 joints: Head, Neck, SpineMid, ShoulderLeft, ElbowLeft, HandLeft, ShoulderRight, ElbowRight, HandRight, HipLeft, KneeLeft, FootLeft, HipRight, KneeRight, and FootRight.

Listing B.2: Python script to pull out the positions of the joints and cameras

```
1  import maya.mel
2  import maya.cmds as cmds
3
4  #FileName = maya.mel.eval("file -q -sn -shn;");
5  #FileName = FileName.replace(".mb", "");
6
7  Character = "Brooke";
8  Action = "Stretch";
9  FileName = Character + "_" + Action;
10
11 for i in range(1, 28):
12         with open("C:/Users/Mashaan_Awad/Documents/maya/projects/default/scripts/
               "+FileName+"_Cameras.txt", "a") as f:
13                 coord = maya.mel.eval('xform -q -t -ws camera'+str(i)+';')
14                 for coordCounter in coord:
15                         f.write("%s " % coordCounter)
16                 f.write("\r\n")
17
18 if Action == "Balance":
```

```
19          startFrame = 1
20          endFrame = 41
21  elif Action == "Jump":
22          startFrame=80
23          endFrame=120
24  elif Action == "Kick":
25          startFrame=1
26          endFrame=41
27  elif Action == "Pickup":
28          startFrame=1
29          endFrame=65
30  elif Action == "Punch":
31          startFrame=1
32          endFrame=30
33  elif Action == "Stretch":
34          startFrame=140
35          endFrame=240
36
37  mel.eval('currentTime %s ;'%(startFrame))
38  while(startFrame <= endFrame):
39          with open("C:/Users/Mashaan_Awad/Documents/maya/projects/default/scripts/
                "+FileName+".txt", "a") as f:
40                  coord = maya.mel.eval("xform -q -t -ws Head ;")
41                  for coordCounter in coord:
42                          f.write("%s " % coordCounter)
43                  coord = maya.mel.eval("xform -q -t -ws Neck ;")
44                  for coordCounter in coord:
45                          f.write("%s " % coordCounter)
46                  coord = maya.mel.eval("xform -q -t -ws Spine1 ;")#Torso
47                  for coordCounter in coord:
48                          f.write("%s " % coordCounter)
49                  coord = maya.mel.eval("xform -q -t -ws LeftArm ;")#Left Shoulder
50                  for coordCounter in coord:
51                          f.write("%s " % coordCounter)
52                  coord = maya.mel.eval("xform -q -t -ws LeftForeArm ;")#left elbow
53                  for coordCounter in coord:
54                          f.write("%s " % coordCounter)
55                  coord = maya.mel.eval("xform -q -t -ws LeftHand ;")#left wrist
56                  for coordCounter in coord:
57                          f.write("%s " % coordCounter)
58                  coord = maya.mel.eval("xform -q -t -ws RightArm ;")#right
                        Shoulder
59                  for coordCounter in coord:
60                          f.write("%s " % coordCounter)
61                  coord = maya.mel.eval("xform -q -t -ws RightForeArm ;")#right
                        elbow
62                  for coordCounter in coord:
63                          f.write("%s " % coordCounter)
64                  coord = maya.mel.eval("xform -q -t -ws RightHand ;")#right wrist
65                  for coordCounter in coord:
66                          f.write("%s " % coordCounter)
67                  coord = maya.mel.eval("xform -q -t -ws LeftUpLeg ;")#left hip
68                  for coordCounter in coord:
69                          f.write("%s " % coordCounter)
70                  coord = maya.mel.eval("xform -q -t -ws LeftLeg ;")#left knee
71                  for coordCounter in coord:
72                          f.write("%s " % coordCounter)
73                  coord = maya.mel.eval("xform -q -t -ws LeftFoot ;")#left foot
74                  for coordCounter in coord:
75                          f.write("%s " % coordCounter)
76                  coord = maya.mel.eval("xform -q -t -ws RightUpLeg ;")#right hip
77                  for coordCounter in coord:
78                          f.write("%s " % coordCounter)
79                  coord = maya.mel.eval("xform -q -t -ws RightLeg ;")#right knee
80                  for coordCounter in coord:
81                          f.write("%s " % coordCounter)
82                  coord = maya.mel.eval("xform -q -t -ws RightFoot ;")#right foot
83                  for coordCounter in coord:
```

```
84                              f.write("%s " % coordCounter)
85                      f.write("\r\n")
86              startFrame += 1
87              mel.eval('currentTime %s ;'%(startFrame))
```

B.7 Segment and Preprocess Joints Positions Exported by Maya

After exporting the world positions of joints and cameras, we subtract each camera position from all joints positions in order to place that camera at the origin. Afterwards, we preprocess the generated samples, the preprocessing contains:

- **Translate:** find the centroid of all coordinates then subtract it from all coordinates to move the player into the middle of the scene (see line 30 to 73 of Listing B.3).
- **Normalize:** all the coordinates divided by the distance between the neck and the spinemid (see line 75 to 79 of Listing B.3).

Listing B.3: Matlab script to Segment and Preprocess Joints Positions Exported by Maya

```
1  % this script reads the world positions of body joints
2  % and camera locations exported by Maya, the subtract eache camera location
3  % from the body joints positions in order to put the camera as global
4  % origin. Also, this script preprocess the samples by performing
5  % translation and normalization
6
7  files = dir('*.txt');
8  NumOfJoints = 15;
9  NumOfDimension = 3;
10 NumOfFeatures = NumOfJoints*NumOfDimension;
11
12 for k=1:2:length(files)
13     WorldCoord = load(files(k).name);
14     WorldCoord = WorldCoord/100;
15     Cameras = load(files(k+1).name);
16     Cameras = Cameras/100;
17     FileName = files(k).name;
18     FileName = strrep(FileName,'.txt','');
19
20     for j=1:size(Cameras,1)
21         CurrentCamera = Cameras(j,:);
22         CurrentInstance = WorldCoord;
23         for i=1:3:size(CurrentInstance,2)
24             %disp(i);
25             CurrentInstance(:,i) = CurrentInstance(:,i) - CurrentCamera(1);
26             CurrentInstance(:,i+1) = CurrentInstance(:,i+1) - CurrentCamera(2);
27             CurrentInstance(:,i+2) = CurrentInstance(:,i+2) - CurrentCamera(3);
28         end
29
30         NumOfFrames = size(CurrentInstance,1);
31         x = zeros(NumOfFrames,(NumOfFeatures/3));
32         y = zeros(NumOfFrames,(NumOfFeatures/3));
33         z = zeros(NumOfFrames,(NumOfFeatures/3));
34         xMean = zeros(NumOfFrames,1);
35         yMean = zeros(NumOfFrames,1);
36         zMean = zeros(NumOfFrames,1);
```

```
37          coordinates1 = zeros(NumOfFrames,NumOfFeatures);
38
39          colCounter=1;
40          for k=1:3:43
41              x(:,colCounter)=CurrentInstance(:,k);
42              colCounter=colCounter+1;
43          end
44          colCounter=1;
45          for k=2:3:44
46              y(:,colCounter)=CurrentInstance(:,k);
47              colCounter=colCounter+1;
48          end
49          colCounter=1;
50          for k=3:3:45
51              z(:,colCounter)=CurrentInstance(:,k);
52              colCounter=colCounter+1;
53          end
54          for k=1:NumOfFrames
55              xMean(k,1)=(sum(x(k,:))/15);
56          end
57          for k=1:NumOfFrames
58              yMean(k,1)=(sum(y(k,:))/15);
59          end
60          for k=1:NumOfFrames
61              zMean(k,1)=(sum(z(k,:))/15);
62          end
63
64          coordinates1 = CurrentInstance;
65          for k=1:3:43
66              coordinates1(:,k) = coordinates1(:,k)-xMean;
67          end
68          for k=2:3:44
69              coordinates1(:,k) = coordinates1(:,k)-yMean;
70          end
71          for k=3:3:45
72              coordinates1(:,k) = coordinates1(:,k)-zMean;
73          end
74
75          xDiff = (CurrentInstance(1,4)-CurrentInstance(1,7))^2;
76          yDiff = (CurrentInstance(1,5)-CurrentInstance(1,8))^2;
77          zDiff = (CurrentInstance(1,6)-CurrentInstance(1,9))^2;
78          dist = sqrt(xDiff+yDiff+zDiff);
79          coordinates1 = coordinates1/dist;
80
81          dlmwrite([FileName '_Camera' num2str(j) '.txt'],coordinates1,'delimiter',
                '\t')
82      end
83  end
```

APPENDIX C

FEATURES EXTRACTION AND CLASSIFICATION

C.1 Features Extraction

We introduced a Matlab script (see Listing C.1) that reads the preprocessed samples produced in the previous sections, and extract features from them. The script produces 15 csv files where each file contains different features combinations alongside their labels. Table C.1 explains the extracted features.

WARNING: Matlab script shown in Listing C.1 should be executed on training and testing samples independently. Running this script on both training and testing samples will merge them into a single csv file which is not desirable in some cases.

Table C.1: Extracted Features

No.	Abbr.	Size	Used Joints	Description
1	Stdev (V1)	45	15	Standard divination of every joint in a single dimension.
2	Euc (V2)	60	15	Maximum 1D Euclidean distance and 3D Euclidean distance between the joint position in the first frame and its position in the middle and last frames.
3	DWTcAStat (V3)	96	4	8 Statistical properties of the DWT approximation coefficients, performed on body extremities trajectories (i.e. Left Hand, Right Hand, Left Foot, and Right Foot).
4	DWTcDStat (V4)	96	4	8 Statistical properties of the DWT detail coefficients, performed on body extremities trajectories (i.e. Left Hand, Right Hand, Left Foot, and Right Foot).
5	V1+V2	105	-	Feature vector #1 combined with Feature vector #2
6	V1+V3	141	-	Feature vector #1 combined with Feature vector #3
7	V1+V4	141	-	Feature vector #1 combined with Feature vector #4
8	V2+V3	156	-	Feature vector #2 combined with Feature vector #3
9	V2+V4	156	-	Feature vector #2 combined with Feature vector #4
10	V3+V4	192	-	Feature vector #3 combined with Feature vector #4
11	V1+V2+V3	201	-	Feature vector #1 combined with Feature vector #2 and Feature vector #3
12	V1+V2+V4	201	-	Feature vector #1 combined with Feature vector #2 and Feature vector #4
13	V1+V3+V4	237	-	Feature vector #1 combined with Feature vector #3 and Feature vector #4
14	V2+V3+V4	252	-	Feature vector #2 combined with Feature vector #3 and Feature vector #4
15	V1+V2+V3+V4	297	-	Feature vector #1 combined with Feature vector #2, Feature vector #3, and Feature vector #4

Listing C.1: Matlab script to Extract Features from Preprocessed Samples

```
MotherWavelet = 'db7';
DatasetName = [MotherWavelet 'RealQuantizedHalf']; %Synthetic Quantized
files = dir('*.txt');
NumOfJoints = 15;
NumOfDimension = 3;
NumOfFeatures = NumOfJoints*NumOfDimension;

Stdev = zeros(length(files),NumOfFeatures+1);
Euc = zeros(length(files),NumOfJoints*2*2+1);
DWTcAStat = zeros(length(files),8*4*NumOfDimension+1);
DWTcDStat = zeros(length(files),8*4*NumOfDimension+1);

V1V2 = zeros(length(files),(size(Stdev,2)-1)+(size(Euc,2)-1)+1);
V1V3 = zeros(length(files),(size(Stdev,2)-1)+(size(DWTcAStat,2)-1)+1);
V1V4 = zeros(length(files),(size(Stdev,2)-1)+(size(DWTcDStat,2)-1)+1);
V2V3 = zeros(length(files),(size(Euc,2)-1)+(size(DWTcAStat,2)-1)+1);
V2V4 = zeros(length(files),(size(Euc,2)-1)+(size(DWTcDStat,2)-1)+1);
V3V4 = zeros(length(files),(size(DWTcAStat,2)-1)+(size(DWTcDStat,2)-1)+1);
V1V2V3 = zeros(length(files),(size(Stdev,2)-1)+(size(Euc,2)-1)+(size(DWTcAStat,2)
    -1)+1);
V1V2V4 = zeros(length(files),(size(Stdev,2)-1)+(size(Euc,2)-1)+(size(DWTcDStat,2)
    -1)+1);
V1V3V4 = zeros(length(files),(size(Stdev,2)-1)+(size(DWTcAStat,2)-1)+(size(
    DWTcDStat,2)-1)+1);
V2V3V4 = zeros(length(files),(size(Euc,2)-1)+(size(DWTcAStat,2)-1)+(size(
    DWTcDStat,2)-1)+1);
V1V2V3V4 = zeros(length(files),(size(Stdev,2)-1)+(size(Euc,2)-1)+(size(DWTcAStat
    ,2)-1)+(size(DWTcDStat,2)-1)+1);

for j=1:length(files)
    coordinates = load(files(j).name);


    NumOfFrames = size(coordinates,1);

    STDEV = zeros(1,NumOfFeatures);
    FullEuc = zeros(1,2*NumOfJoints);
    MidEuc = zeros(1,2*NumOfJoints);

    coordinates1 = coordinates;


    % stores the STDEV for the joint positions over time
    for k=1:45
        STDEV(k) = std(coordinates1(:,k));
    end

    % stores 3D Euc distance and maximum 1D distance for (first,middle) and
    % (first,last)
    EucCounter = 1;
    for k=1:3:NumOfFeatures
        % first frame
        first = coordinates1(1,k:k+2);
        % middle frame
        mid = coordinates1(round(size(coordinates1,1)/2),k:k+2);
        % last frame
        last = coordinates1(size(coordinates1,1),k:k+2);
        % stores 3D Euc distance between first and middle frame
        MidEuc(EucCounter) = pdist2(mid,first);
        % stores maximum 1D Euc distance between first and middle frame
        MidEuc(EucCounter+1) = max(abs(mid-first));
```

```
57          % stores 3D Euc distance between first and last frame
58          FullEuc(EucCounter) = pdist2(last,first);
59          % stores maximum 1D Euc distance between first and last frame
60          FullEuc(EucCounter+1) = max(abs(last-first));
61          EucCounter = EucCounter+2;
62      end
63
64      LeftHand = coordinates1(:,16:18)';
65      LeftHandcA = [];
66      LeftHandcD = [];
67      LeftHandcAStat = [];
68      LeftHandcDStat = [];
69      RightHand = coordinates1(:,25:27)';
70      RightHandcA = [];
71      RightHandcD = [];
72      RightHandcAStat = [];
73      RightHandcDStat = [];
74      LeftFoot = coordinates1(:,34:36)';
75      LeftFootcA = [];
76      LeftFootcD = [];
77      LeftFootcAStat = [];
78      LeftFootcDStat = [];
79      RightFoot = coordinates1(:,43:45)';
80      RightFootcA = [];
81      RightFootcD = [];
82      RightFootcAStat = [];
83      RightFootcDStat = [];
84
85      for k=1:3
86          [cA,cD] = dwt(LeftHand(k,:),MotherWavelet);
87          LeftHandcA = [LeftHandcA; cA];
88          LeftHandcD = [LeftHandcD; cD];
89      end
90      for k=1:3
91          temp1 = [std(LeftHandcA(k,:)) max(LeftHandcA(k,:)) min(LeftHandcA(k,:))
                quantile(LeftHandcA(k,:),0.25) quantile(LeftHandcA(k,:),0.50)
                quantile(LeftHandcA(k,:),0.75) skewness(LeftHandcA(k,:)) kurtosis(
                LeftHandcA(k,:))];
92          temp2 = [std(LeftHandcD(k,:)) max(LeftHandcD(k,:)) min(LeftHandcD(k,:))
                quantile(LeftHandcD(k,:),0.25) quantile(LeftHandcD(k,:),0.50)
                quantile(LeftHandcD(k,:),0.75) skewness(LeftHandcD(k,:)) kurtosis(
                LeftHandcD(k,:))];
93          LeftHandcAStat = [LeftHandcAStat temp1];
94          LeftHandcDStat = [LeftHandcDStat temp2];
95      end
96
97      for k=1:3
98          [cA,cD] = dwt(RightHand(k,:),MotherWavelet);
99          RightHandcA = [RightHandcA; cA];
100         RightHandcD = [RightHandcD; cD];
101     end
102     for k=1:3
103         temp1 = [std(RightHandcA(k,:)) max(RightHandcA(k,:)) min(RightHandcA(k,:)
                ) quantile(RightHandcA(k,:),0.25) quantile(RightHandcA(k,:),0.50)
                quantile(RightHandcA(k,:),0.75) skewness(RightHandcA(k,:)) kurtosis(
                RightHandcA(k,:))];
104         temp2 = [std(RightHandcD(k,:)) max(RightHandcD(k,:)) min(RightHandcD(k,:)
                ) quantile(RightHandcD(k,:),0.25) quantile(RightHandcD(k,:),0.50)
                quantile(RightHandcD(k,:),0.75) skewness(RightHandcD(k,:)) kurtosis(
                RightHandcD(k,:))];
105         RightHandcAStat = [RightHandcAStat temp1];
106         RightHandcDStat = [RightHandcDStat temp2];
107     end
108
109     for k=1:3
110         [cA,cD] = dwt(LeftFoot(k,:),MotherWavelet);
111         LeftFootcA = [LeftFootcA; cA];
112         LeftFootcD = [LeftFootcD; cD];
```

```
113     end
114     for k=1:3
115         temp1 = [std(LeftFootcA(k,:)) max(LeftFootcA(k,:)) min(LeftFootcA(k,:))
                quantile(LeftFootcA(k,:),0.25) quantile(LeftFootcA(k,:),0.50)
                quantile(LeftFootcA(k,:),0.75) skewness(LeftFootcA(k,:)) kurtosis(
                LeftFootcA(k,:))];
116         temp2 = [std(LeftFootcD(k,:)) max(LeftFootcD(k,:)) min(LeftFootcD(k,:))
                quantile(LeftFootcD(k,:),0.25) quantile(LeftFootcD(k,:),0.50)
                quantile(LeftFootcD(k,:),0.75) skewness(LeftFootcD(k,:)) kurtosis(
                LeftFootcD(k,:))];
117         LeftFootcAStat = [LeftFootcAStat temp1];
118         LeftFootcDStat = [LeftFootcDStat temp2];
119     end
120
121     for k=1:3
122         [cA,cD] = dwt(RightFoot(k,:),MotherWavelet);
123         RightFootcA = [RightFootcA; cA];
124         RightFootcD = [RightFootcD; cD];
125     end
126     for k=1:3
127         temp1 = [std(RightFootcA(k,:)) max(RightFootcA(k,:)) min(RightFootcA(k,:)
                ) quantile(RightFootcA(k,:),0.25) quantile(RightFootcA(k,:),0.50)
                quantile(RightFootcA(k,:),0.75) skewness(RightFootcA(k,:)) kurtosis(
                RightFootcA(k,:))];
128         temp2 = [std(RightFootcD(k,:)) max(RightFootcD(k,:)) min(RightFootcD(k,:)
                ) quantile(RightFootcD(k,:),0.25) quantile(RightFootcD(k,:),0.50)
                quantile(RightFootcD(k,:),0.75) skewness(RightFootcD(k,:)) kurtosis(
                RightFootcD(k,:))];
129         RightFootcAStat = [RightFootcAStat temp1];
130         RightFootcDStat = [RightFootcDStat temp2];
131     end
132
133     if ~(isempty(strfind(files(j).name, 'Balance')))
134         label = 1;
135     elseif ~(isempty(strfind(files(j).name, 'Jump')))
136         label = 2;
137     elseif ~(isempty(strfind(files(j).name, 'Kick')))
138         label = 3;
139     elseif ~(isempty(strfind(files(j).name, 'Pickup')))
140         label = 4;
141     elseif ~(isempty(strfind(files(j).name, 'Punch')))
142         label = 5;
143     elseif ~(isempty(strfind(files(j).name, 'Stretch')))
144         label = 6;
145     end
146
147     TempDWTcAStat = [LeftHandcAStat RightHandcAStat LeftFootcAStat
            RightFootcAStat];
148     TempDWTcDStat = [LeftHandcDStat RightHandcDStat LeftFootcDStat
            RightFootcDStat];
149
150     Stdev(j,:) = [STDEV label];
151     Euc(j,:) = [FullEuc MidEuc label];
152     DWTcAStat(j,:) = [TempDWTcAStat label];
153     DWTcDStat(j,:) = [TempDWTcDStat label];
154     V1V2(j,:) = [STDEV FullEuc MidEuc label];
155     V1V3(j,:) = [STDEV TempDWTcAStat label];
156     V1V4(j,:) = [STDEV TempDWTcDStat label];
157     V2V3(j,:) = [FullEuc MidEuc TempDWTcAStat label];
158     V2V4(j,:) = [FullEuc MidEuc TempDWTcDStat label];
159     V3V4(j,:) = [TempDWTcAStat TempDWTcDStat label];
160     V1V2V3(j,:) = [STDEV FullEuc MidEuc TempDWTcAStat label];
161     V1V2V4(j,:) = [STDEV FullEuc MidEuc TempDWTcDStat label];
162     V1V3V4(j,:) = [STDEV TempDWTcAStat TempDWTcDStat label];
163     V2V3V4(j,:) = [FullEuc MidEuc TempDWTcAStat TempDWTcDStat label];
164     V1V2V3V4(j,:) = [STDEV FullEuc MidEuc TempDWTcAStat TempDWTcDStat label];
165 end
166
```

```
167 csvwrite([DatasetName 'Stdev.csv'],[1:size(Stdev,2); Stdev]);
168 csvwrite([DatasetName 'Euc.csv'],[1:size(Euc,2); Euc]);
169 csvwrite([DatasetName 'DWTcAStat.csv'],[1:size(DWTcAStat,2); DWTcAStat]);
170 csvwrite([DatasetName 'DWTcDStat.csv'],[1:size(DWTcDStat,2); DWTcDStat]);
171 csvwrite([DatasetName 'V1V2.csv'],[1:size(V1V2,2); V1V2]);
172 csvwrite([DatasetName 'V1V3.csv'],[1:size(V1V3,2); V1V3]);
173 csvwrite([DatasetName 'V1V4.csv'],[1:size(V1V4,2); V1V4]);
174 csvwrite([DatasetName 'V2V3.csv'],[1:size(V2V3,2); V2V3]);
175 csvwrite([DatasetName 'V2V4.csv'],[1:size(V2V4,2); V2V4]);
176 csvwrite([DatasetName 'V3V4.csv'],[1:size(V3V4,2); V3V4]);
177 csvwrite([DatasetName 'V1V2V3.csv'],[1:size(V1V2V3,2); V1V2V3]);
178 csvwrite([DatasetName 'V1V2V4.csv'],[1:size(V1V2V4,2); V1V2V4]);
179 csvwrite([DatasetName 'V1V3V4.csv'],[1:size(V1V3V4,2); V1V3V4]);
180 csvwrite([DatasetName 'V2V3V4.csv'],[1:size(V2V3V4,2); V2V3V4]);
181 csvwrite([DatasetName 'V1V2V3V4.csv'],[1:size(V1V2V3V4,2); V1V2V3V4]);
```

C.2 Convert CSV files to ARFF files

From the previous subsection we obtain a csv files with labeled instances. However, to perform the classification using Weka we need to convert csv files into Attribute-Relation File Format (ARFF). This step can be done using Weka Explorer, though it might be difficult for multiple files therefore we need automation. For csv files we need to perform the following steps:

- Specify the class attribute and convert it to nominal.
- Convert it to ARFF.

The Java code shown in Listing C.2 locates the csv files produced by Matlab (change the directory in line 17) and convert them into ARFF files. Remeber you need Weka Java API to run the Java code in Listing C.2, in other words you need to include (Weka.jar) in your java project dependencies.

Listing C.2: Java code to convert multiple csv files into ARFF files

```
1  import java.io.File;
2  import java.io.FilenameFilter;
3  import java.io.IOException;
4  import java.util.Arrays;
5  import java.util.List;
6
7  import weka.core.Instances;
8  import weka.core.converters.ArffSaver;
9  import weka.core.converters.CSVLoader;
10 import weka.filters.Filter;
11 import weka.filters.unsupervised.attribute.NumericToNominal;
12
13
14 public class CSV2Arff {
15
16         public static void main(String[] args) throws Exception {
17                 File dir = new File("C:/Users/Mashaan_Awad/Documents/MATLAB/Data
                       Synthesized/Classification Results");
18                 File[] listFiles = dir.listFiles();
19
20                 for(int i=0; i<listFiles.length;i++)
21                 {
22                         System.out.println(i+"-"+listFiles[i]);
23                         String filename = listFiles[i].getName();
```

```
24                          if (filename.contains("csv")) {
25                                  if (filename.indexOf(".") > 0) {
26                                      filename = filename.substring(0, filename.
                                            lastIndexOf("."));
27                                  }
28                                  System.out.println(i+"-"+filename);
29
30                                  // load CSV
31                              CSVLoader loader = new CSVLoader();
32                              loader.setSource(listFiles[i]);
33                              Instances data = loader.getDataSet();
34                              data.setClassIndex(data.numAttributes()-1);
35                              NumericToNominal Nominalize= new NumericToNominal();
36                              String[] options= new String[2];
37                              options[0] = "-R";
38                              options[1] = "last"; //set the attributes from
                                    indices 1 to 2 as
39
40                              //nominal
41                              Nominalize.setOptions(options);
42                              Nominalize.setInputFormat(data);
43                              Instances data1 = Filter.useFilter(data, Nominalize);
44
45                                  // save ARFF
46                              ArffSaver saver = new ArffSaver();
47                              saver.setInstances(data1);
48                              saver.setFile(new File(dir+"/"+filename+".arff"));
49                              saver.writeBatch();
50                          }
51                  }
52          }
53
54  }
```

C.3 Classify Instances using Weka Java API

After converting csv files into ARFF files, we are ready to perform the classification. Like the previous subsection, this process can be performed using Weka Explorer, however automating this process for multiple datasets will save time and effort. Code in Listing C.3 reads multiple datasets specified by the directory in line 14. Then the user should fill (DatasetsNames) array with the datasets names. First, the training set should be written and then followed by its testing dataset. The code executes by train on the first dataset and make predictions on the following dataset, this will be repeated until all datasets in (DatasetsNames) got processed. The code in Listing C.3 uses random forest classifier (see line 51), though any other classifier supported by Weka can be used.

Listing C.3: Java code to classify multiple ARFF files

```
1  import java.io.File;
2  import java.util.Random;
3
4  import weka.classifiers.Evaluation;
5  import weka.classifiers.functions.SMO;
6  import weka.classifiers.trees.RandomForest;
7  import weka.core.Instances;
8  import weka.core.converters.ArffLoader;
9  import weka.core.converters.ConverterUtils.DataSource;
10
```

```
11 public class TrainAndTest {
12
13         public static void main(String[] args) throws Exception {
14                 String dir = new String("C:/Users/Mashaan_Awad/Documents/MATLAB/
                       Data Synthesized/Classification Results/");
15                 String DatasetsNames[] = {"SyntheticPreprocessedDWTcAcDStat","
                       RealPreprocessedDWTcAcDStat",
16                                 "SyntheticPreprocessedDWTcDStat","
                                       RealPreprocessedDWTcDStat",
17                                 "SyntheticPreprocessedEuc","RealPreprocessedEuc",
18                                 "SyntheticPreprocessedEucDWTcAcDStat","
                                       RealPreprocessedEucDWTcAcDStat",
19                                 "SyntheticPreprocessedEucDWTcDStat","
                                       RealPreprocessedEucDWTcDStat",
20                                 "SyntheticPreprocessedQuantizedDWTcAcDStat","
                                       RealPreprocessedQuantizedDWTcAcDStat",
21                                 "SyntheticPreprocessedQuantizedDWTcDStat","
                                       RealPreprocessedQuantizedDWTcDStat",
22                                 "SyntheticPreprocessedQuantizedEuc","
                                       RealPreprocessedQuantizedEuc",
23                                 "SyntheticPreprocessedQuantizedEucDWTcAcDStat","
                                       RealPreprocessedQuantizedEucDWTcAcDStat",
24                                 "SyntheticPreprocessedQuantizedEucDWTcDStat","
                                       RealPreprocessedQuantizedEucDWTcDStat",
25                                 "SyntheticPreprocessedQuantizedStdev","
                                       RealPreprocessedQuantizedStdev",
26                                 "SyntheticPreprocessedQuantizedStdevDWTcAcDStat",
                                       "RealPreprocessedQuantizedStdevDWTcAcDStat",
27                                 "SyntheticPreprocessedQuantizedStdevDWTcDStat","
                                       RealPreprocessedQuantizedStdevDWTcDStat",
28                                 "SyntheticPreprocessedQuantizedStdevEuc","
                                       RealPreprocessedQuantizedStdevEuc",
29                                 "
                                       SyntheticPreprocessedQuantizedStdevEucDWTcAcDStat
                                       ","
                                       RealPreprocessedQuantizedStdevEucDWTcAcDStat"
                                       ,
30                                 "SyntheticPreprocessedQuantizedStdevEucDWTcDStat"
                                       ,"RealPreprocessedQuantizedStdevEucDWTcDStat"
                                       ,
31                                 "SyntheticPreprocessedStdev","
                                       RealPreprocessedStdev",
32                                 "SyntheticPreprocessedStdevDWTcAcDStat","
                                       RealPreprocessedStdevDWTcAcDStat",
33                                 "SyntheticPreprocessedStdevDWTcDStat","
                                       RealPreprocessedStdevDWTcDStat",
34                                 "SyntheticPreprocessedStdevEuc","
                                       RealPreprocessedStdevEuc",
35                                 "SyntheticPreprocessedStdevEucDWTcAcDStat","
                                       RealPreprocessedStdevEucDWTcAcDStat",
36                                 "SyntheticPreprocessedStdevEucDWTcDStat","
                                       RealPreprocessedStdevEucDWTcDStat"};
37
38                 for(int i=0; i<DatasetsNames.length;i+=2)
39                 //for(int i=0; i<2;i+=2)
40                 {
41                         System.out.println("Train Dataset = "+DatasetsNames[i]);
42                         System.out.println("Test Dataset = "+DatasetsNames[i+1]);
43                         DataSource TrainSource = new DataSource(dir+DatasetsNames
                               [i]+".arff");
44                         Instances Train = TrainSource.getDataSet();
45                         Train.setClassIndex(Train.numAttributes() - 1);
46
47                         DataSource TestSource = new DataSource(dir+DatasetsNames[
                               i+1]+".arff");
48                         Instances Test = TestSource.getDataSet();
49                         Test.setClassIndex(Train.numAttributes() - 1);
50
```

```
51                          RandomForest RF = new RandomForest();
52                          RF.buildClassifier(Train);
53              Evaluation eval = new Evaluation(Train);
54              eval.evaluateModel(RF, Test);
55              System.out.println("Train Instances = " + Train.numInstances()+",
                    Test Instances = " + Test.numInstances());
56                          System.out.println("Train Attributes = " + Train.
                                numAttributes()+", Test Attributes = " + Test.
                                numAttributes());
57                          System.out.println("Estimated Accuracy: "+Double.toString
                                (eval.pctCorrect()));
58                          System.out.print("Precision ordered by class: ");
59                          for(int j=0; j<Test.numClasses();j++)
60                          {
61                                  System.out.format("%.4f, ",eval.precision(j));
62                          }
63                          System.out.println();
64                          System.out.print("Recall ordered by class: ");
65                          for(int j=0; j<Test.numClasses();j++)
66                          {
67                                  System.out.format("%.4f, ",eval.recall(j));
68                          }
69                          System.out.println();
70                          System.out.print("FMeasure ordered by class: ");
71                          for(int j=0; j<Test.numClasses();j++)
72                          {
73                                  System.out.format("%.4f, ",eval.fMeasure(j));
74                          }
75                          System.out.println();
76                          System.out.println("
                                ==========================================================");
77                  }
78          }
79
80  }
```

APPENDIX D

MISCELLANEOUS

D.1 Draw a stick model

A stick model is a collection of points connected by lines where the points are the joints and the lines are the bones (see Figure D.1). The text files generated in real data and synthetic data have the same convention where the number of columns is 45 considering 15 joints and 3 dimensions. The number of rows varies from sample to sample because it represents the number of frames. The Matlab script shown in Listing D.1 reads multiple text files and generates a plot of the stick model for each frame.

WARNING: The joints in the text file should be order as follows: Head, Neck, SpineMid, ShoulderLeft, ElbowLeft, HandLeft, ShoulderRight, ElbowRight, HandRight, HipLeft, KneeLeft, FootLeft, HipRight, KneeRight, and FootRight. Otherwise, the script will plot a faulty stick model. Anyway, you can control the joints order by modifying the lines 14 to 28 of Listing D.1.

Listing D.1: Matlab script to Draw Stick Model

```
1  files = dir('*.txt');
2
3  for j=1:length(files)
4      close all;
5
6      coord = load(files(j).name);
7      %# preallocate
8      nFrames = size(coord,1);
9      f = getframe(gca);
10     [f,map] = rgb2ind(f.cdata, 256, 'nodither');
11     mov = repmat(f, [1 1 1 nFrames]);
12
13     for i=1:size(coord,1)
14         Head = coord(i,1:3);
15         Neck = coord(i,4:6);
16         Torso = coord(i,7:9);
17         ShoulderLeft = coord(i,10:12);
18         ElbowLeft = coord(i,13:15);
19         HandLeft = coord(i,16:18);
20         ShoulderRight = coord(i,19:21);
```

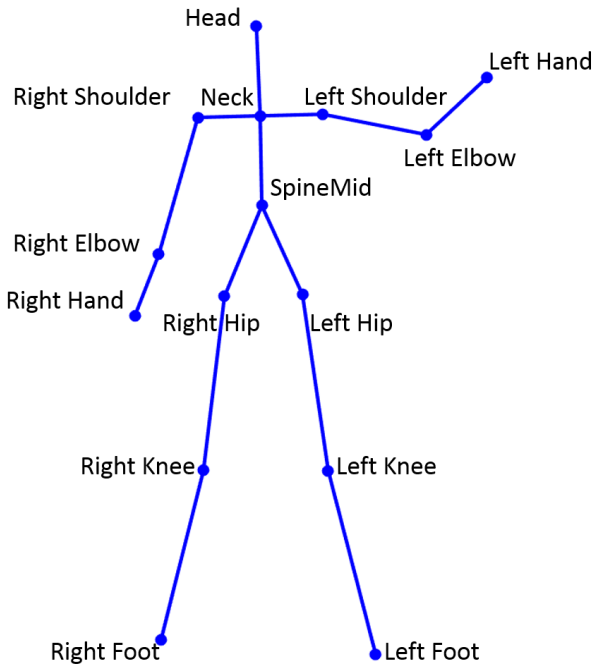


Figure D.1: Stick Model

```
21          ElbowRight = coord(i,22:24);
22          HandRight = coord(i,25:27);
23          HipLeft = coord(i,28:30);
24          KneeLeft = coord(i,31:33);
25          FootLeft = coord(i,34:36);
26          HipRight = coord(i,37:39);
27          KneeRight = coord(i,40:42);
28          FootRight = coord(i,43:45);
29
30          coord1 = [Head; Neck; Torso; ShoulderLeft; ElbowLeft; HandLeft;
                ShoulderRight; ElbowRight; HandRight; HipLeft; KneeLeft; FootLeft;
                HipRight; KneeRight; FootRight]
31
32          BackBone = [Head; Neck; Torso];
33          LeftArm = [Neck; ShoulderLeft; ElbowLeft; HandLeft];
34          RightArm = [Neck; ShoulderRight; ElbowRight; HandRight];
35          LeftLeg = [Torso; HipLeft; KneeLeft; FootLeft];
36          RightLeg = [Torso; HipRight; KneeRight; FootRight];
37
38          cla;
39          plot3(coord1(:,1), coord1(:,2), coord1(:,3), 'bo');
40          set(gcf, 'color', [1 1 1]);
41          set(gca, 'Visible','off');
42          %view([-20 50]);
43          %view([180 270]);
44          view(2);
45          xlabel('x-axis') % x-axis label
46          ylabel('y-axis') % y-axis label
47          ylabel('z-axis') % z-axis label
48          axis equal;
49          hold on;
50          line(BackBone(:,1),BackBone(:,2),BackBone(:,3));
51          line(LeftArm(:,1),LeftArm(:,2),LeftArm(:,3));
52          line(RightArm(:,1),RightArm(:,2),RightArm(:,3));
53          line(LeftLeg(:,1),LeftLeg(:,2),LeftLeg(:,3));
54          line(RightLeg(:,1),RightLeg(:,2),RightLeg(:,3));
```

```
55          f = getframe(gca);
56          mov(:,:,1,i) = rgb2ind(f.cdata, map, 'nodither');
57      end
58
59      close(gcf)
60      imwrite(mov, map, [strrep(files(j).name, '.txt', '') '.gif'], 'DelayTime'
            ,0.1, 'LoopCount',inf);
61  end
```

D.2 Create GIF image

We can animate the stick model by combining all frames of a single text file (i.e., single action) into single GIF image. Matlab script shown in Listing D.2, illustrates the process of generating GIF image.

WARNING: The text files should be numbered from 1 to N where N is total number of frames. Also, note that Matlab might order the files wrongly. For example, Matlab orderers (1, 2, 10) as (1, 10, 2), therefore you might need to stuff some zeros to force Matlab to get it right such as (0001, 0002, 0010).

Listing D.2: Matlab script to Generate GIF Image

```
1  files = dir('*.png');
2  FileName = files(1).name;
3  FileName = strrep(FileName,'_1.png','');
4
5  for j=1:length(files)
6      image = imread([FileName '_' num2str(j) '.png'], 'BackgroundColor',[1 1 1]);
7      [A,map] = rgb2ind(image,256);
8      if (j==1)
9          imwrite(A,map,[FileName '.gif'],'gif','LoopCount',Inf,'DelayTime',0.1);
10     else
11         imwrite(A,map,[FileName '.gif'],'gif','WriteMode','append','DelayTime'
               ,0.1);
12     end
13 end
```

REFERENCES

[1] MathWorks, "MATLAB® Documentation," http://www.mathworks.com/help/matlab/, accessed: 10 Apr. 2015.

[2] R. Ibañez, Á. Soria, A. Teyseyre, and M. Campo, "Easy gesture recognition for kinect," *Advances in Engineering Software*, vol. 76, no. 0, pp. 171 – 180, 2014.

[3] C. Attolico, G. Cicirelli, C. Guaragnella, and T. DOrazio, "A real time gesture recognition system for human computer interaction," in *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*, ser. Lecture Notes in Computer Science, F. Schwenker, S. Scherer, and L.-P. Morency, Eds. Springer International Publishing, 2015, vol. 8869, pp. 92–101.

[4] C. Sun, T. Zhang, and C. Xu, "Latent support vector machine modeling for sign language recognition with kinect," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 2, pp. 20:1–20:20, Mar. 2015.

[5] S. Saha, M. Pal, A. Konar, and D. Bhattacharya, "Automatic gesture recog-

nition for health care using relieff and fuzzy knn." Springer India, 2015, vol. 340, pp. 709–717.

[6] C. Lea, G. Hager, and R. Vidal, "An improved model for segmentation and recognition of fine-grained activities with application to surgical training tasks," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, Jan 2015, pp. 1123–1129.

[7] I. Laptev, "On space-time interest points," *International Journal of Computer Vision*, vol. 64, no. 2-3, pp. 107–123, 2005.

[8] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3, Aug 2004, pp. 32–36 Vol.3.

[9] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, June 2008, pp. 1–8.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, June 2005, pp. 886–893 vol. 1.

[11] J. Wang, Z. Liu, Y. Wu, and J. Yuan, "Learning actionlet ensemble for 3d human action recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 5, pp. 914–927, May 2014.

[12] L. Chen, H. Wei, and J. Ferryman, "A survey of human motion analysis using depth imagery," *Pattern Recognition Letters*, vol. 34, no. 15, pp. 1995 – 2006, 2013, smart Approaches for Human Action Recognition.

[13] M. Ahad, J. Tan, H. Kim, and S. Ishikawa, "Motion history image: its variants and applications," *Machine Vision and Applications*, vol. 23, no. 2, pp. 255–281, 2012.

[14] M.-C. Roh, H.-K. Shin, and S.-W. Lee, "View-independent human action recognition with volume motion template on single stereo camera," *Pattern Recognition Letters*, vol. 31, no. 7, pp. 639 – 647, 2010.

[15] H. Wang, J. Fu, Y. Lu, X. Chen, and S. Li, "Depth sensor assisted real-time gesture recognition for interactive presentation," *Journal of Visual Communication and Image Representation*, vol. 24, no. 8, pp. 1458 – 1468, 2013.

[16] B. Ni, Y. Pei, P. Moulin, and S. Yan, "Multilevel depth and image fusion for human activity detection," *Cybernetics, IEEE Transactions on*, vol. 43, no. 5, pp. 1383–1394, Oct 2013.

[17] J. Luo, W. Wang, and H. Qi, "Spatio-temporal feature extraction and representation for rgb-d human action recognition," *Pattern Recognition Letters*, vol. 50, pp. 139 – 148, 2014, depth Image Analysis.

[18] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake, "Efficient human

pose estimation from single depth images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 12, pp. 2821–2840, Dec 2013.

[19] T. Arici, S. Celebi, A. Aydin, and T. Temiz, "Robust gesture recognition using feature pre-processing and weighted dynamic time warping," *Multimedia Tools and Applications*, vol. 72, no. 3, pp. 3045–3062, 2014.

[20] H.-S. Le, N.-Q. Pham, and D.-D. Nguyen, "Neural networks with hidden markov models in skeleton-based gesture recognition," in *Knowledge and Systems Engineering*, ser. Advances in Intelligent Systems and Computing, V.-H. Nguyen, A.-C. Le, and V.-N. Huynh, Eds. Springer International Publishing, 2015, vol. 326, pp. 299–311.

[21] M. A. Gowayyed, M. Torki, M. E. Hussein, and M. El-Saban, "Histogram of oriented displacements (hod): Describing trajectories of human joints for action recognition," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '13. AAAI Press, 2013, pp. 1351–1357.

[22] A. Sharaf, M. Torki, M. Hussein, and M. El-Saban, "Real-time multi-scale action detection from 3d skeleton data," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, Jan 2015, pp. 998–1005.

[23] S. M. Chang, "Using gesture recognition to control powerpoint using the microsoft kinect," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[24] D.-L. Dinh, J. T. Kim, and T.-S. Kim, "Hand gesture recognition and interface via a depth imaging sensor for smart home appliances," *Energy Procedia*, vol. 62, no. 0, pp. 576 – 582, 2014, 6th International Conference on Sustainability in Energy and Buildings, SEB-14.

[25] Microsoft, "Kinect® for Windows," http://www.microsoft.com/en-us/kinectforwindows/, accessed: 10 Jul. 2015.

[26] E. Ramos, "Kinect basics," in *Arduino and Kinect Projects*. Apress, 2012, pp. 23–34.

[27] Guinness World Records, "Fastest-selling Gaming Peripheral," http://www.guinnessworldrecords.com/world-records/fastest-selling-gaming-peripheral, accessed: 10 Apr. 2015.

[28] A. Bellucci, A. Malizia, P. Diaz, and I. Aedo, "Human-display interaction technology: Emerging remote interfaces for pervasive display environments," *Pervasive Computing, IEEE*, vol. 9, no. 2, pp. 72–76, April 2010.

[29] Microsoft. The microsoft developer network. [Online]. Available: msdn.microsoft.com

[30] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, fourth edition ed. Boston: Academic Press, 2009.

[31] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[32] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[33] V. sen Feng and S. Y. Chang, "Determination of wireless networks parameters through parallel hierarchical support vector machines," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 3, pp. 505–512, March 2012.

[34] A. Prinzie and D. V. den Poel, "Random forests for multiclass classification: Random multinomial logit," *Expert Systems with Applications*, vol. 34, no. 3, pp. 1721 – 1732, 2008.

[35] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Comput.*, vol. 9, no. 7, pp. 1545–1588, Oct. 1997.

[36] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[37] E. Fix and J. L. Hodges Jr, "Discriminatory analysis-nonparametric discrimination: consistency properties," DTIC Document, Tech. Rep., 1951.

[38] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, January 1967.

[39] L. Devroye, "On the inequality of cover and hart in nearest neighbor discrimination," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-3, no. 1, pp. 75–78, Jan 1981.

[40] Q. Gu, L. Zhu, and Z. Cai, "Evaluation measures of the classification performance of imbalanced data sets," in *Computational Intelligence and Intelligent Systems*, ser. Communications in Computer and Information Science, Z. Cai, Z. Li, Z. Kang, and Y. Liu, Eds. Springer Berlin Heidelberg, 2009, vol. 51, pp. 461–471.

[41] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing Management*, vol. 45, no. 4, pp. 427 – 437, 2009.

[42] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, Mar. 2003.

[43] T. Morris, "Gesture recognition," in *Multimedia Systems*, ser. Applied Computing. Springer US, 2000, pp. 121–136.

[44] B. A. Myers, "A brief history of human-computer interaction technology," *interactions*, vol. 5, no. 2, pp. 44–54, Mar. 1998.

[45] P. Maes, T. Darrell, B. Blumberg, and A. Pentland, "The alive system: full-body interaction with autonomous agents," in *Computer Animation '95., Proceedings.*, Apr 1995, pp. 11–18, 209.

[46] A. Camurri and P. Ferrentino, "Interactive environments for music and multimedia," *Multimedia Systems*, vol. 7, no. 1, pp. 32–47, 1999.

[47] A. Bobick and J. Davis, "The recognition of human movement using temporal

templates," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 3, pp. 257–267, Mar 2001.

[48] Sony Computer Entertainment, "PlayStation Move motion controller," https://www.playstation.com/en-us/explore/accessories/playstation-move/, accessed: 10 Jul. 2015.

[49] R. Poppe, "Vision-based human motion analysis: An overview," *Computer Vision and Image Understanding*, vol. 108, no. 1-2, pp. 4–18, 2007.

[50] Sony Computer Entertainment, "PlayStation EyeToy® camera," http://us.playstation.com/ps2/accessories/eyetoy-usb-camera-ps2.html, accessed: 10 Jul. 2015.

[51] F. Durand, "A short introduction to computer graphics," 2005.

[52] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *Cybernetics, IEEE Transactions on*, vol. 43, no. 5, pp. 1318–1334, Oct 2013.

[53] J. Fabian, T. Young, J. Peyton Jones, and G. Clayton, "Integrating the microsoft kinect with simulink: Real-time object tracking example," *Mechatronics, IEEE/ASME Transactions on*, vol. 19, no. 1, pp. 249–257, Feb 2014.

[54] T. Nakamura, "Real-time 3-d object tracking using kinect sensor," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, Dec 2011, pp. 784–788.

[55] A. Janoch, S. Karayev, Y. Jia, J. Barron, M. Fritz, K. Saenko, and T. Darrell, "A category-level 3d object dataset: Putting the kinect to work," in *Consumer Depth Cameras for Computer Vision*, ser. Advances in Computer Vision and Pattern Recognition, A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, Eds. Springer London, 2013, pp. 141–165.

[56] L. Bo, X. Ren, and D. Fox, "Depth kernel descriptors for object recognition," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, Sept 2011, pp. 821–826.

[57] S. Tang, X. Wang, X. Lv, T. Han, J. Keller, Z. He, M. Skubic, and S. Lao, "Histogram of oriented normal vectors for object recognition with a depth sensor," in *Computer Vision ACCV 2012*, ser. Lecture Notes in Computer Science, K. Lee, Y. Matsushita, J. Rehg, and Z. Hu, Eds. Springer Berlin Heidelberg, 2013, vol. 7725, pp. 525–538.

[58] J. Shotton, S. Izadi, O. Hilliges, D. Kim, D. Molyneaux, M. Cook, P. Kohli, A. Criminisi, R. Girshick, and A. Fitzgibbon, "Human body pose estimation," Jan. 28 2014, uS Patent 8,638,985. [Online]. Available: https://www.google.com/patents/US8638985

[59] S. Zhang, Y. Hui, J. Dong, T. Wang, L. Qi, and H. Liu, "Combining kinect and pnp for camera pose estimation," in *Human System Interactions (HSI), 2015 8th International Conference on*, June 2015, pp. 357–361.

[60] B. Junxia, Y. Jianqin, W. Jun, and Z. Ling, "Hand detection based on depth

information and color information of the kinect," in *Control and Decision Conference (CCDC), 2015 27th Chinese*, May 2015, pp. 4205–4210.

[61] G. Rogez, M. Khademi, J. Supani III, J. Montiel, and D. Ramanan, "3d hand pose detection in egocentric rgb-d images," in *Computer Vision - ECCV 2014 Workshops*, ser. Lecture Notes in Computer Science, L. Agapito, M. M. Bronstein, and C. Rother, Eds. Springer International Publishing, 2015, vol. 8925, pp. 356–371.

[62] Y. Wang, Q. Zhang, and Y. Zhou, "Dense 3d mapping for indoor environment based on kinect-style depth cameras," in *Robot Intelligence Technology and Applications 3*, ser. Advances in Intelligent Systems and Computing, J.-H. Kim, W. Yang, J. Jo, P. Sincak, and H. Myung, Eds. Springer International Publishing, 2015, vol. 345, pp. 317–330.

[63] Q. Zhang, B. Li, G. qing Xu, Y. Zhou, M. Wang, and M.-H. Meng, "Indoor environment applications for mobile robots using kinect2.0," in *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, June 2014, pp. 1462–1466.

[64] M. Devanne, H. Wannous, S. Berretti, P. Pala, M. Daoudi, and A. Del Bimbo, "3-d human action recognition by shape analysis of motion trajectories on riemannian manifold," *Cybernetics, IEEE Transactions on*, vol. 45, no. 7, pp. 1340–1352, July 2015.

[65] T. Zhang and Z. Feng, "Dynamic gesture recognition based on fusing frame

images," in *Intelligent Systems Design and Engineering Applications, 2013 Fourth International Conference on*, Nov 2013, pp. 280–283.

[66] D.-L. Dinh, S. Lee, and T.-S. Kim, "Hand number gesture recognition using recognized hand parts in depth images," *Multimedia Tools and Applications*, pp. 1–16, 2014.

[67] C. Keskin, F. Kra, Y. Kara, and L. Akarun, "Real time hand pose estimation using depth sensors," in *Consumer Depth Cameras for Computer Vision*, ser. Advances in Computer Vision and Pattern Recognition, A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, Eds. Springer London, 2013, pp. 119–137.

[68] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[69] Anupam, K. Biswas, A. Shukla, and R. Pandey, "Depth map based recognition of activities involving two persons," in *Contemporary Computing*, ser. Communications in Computer and Information Science, M. Parashar, D. Kaushik, O. Rana, R. Samtaney, Y. Yang, and A. Zomaya, Eds. Springer Berlin Heidelberg, 2012, vol. 306, pp. 492–493.

[70] S. Saha, S. Datta, A. Konar, and R. Janarthanan, "A study on emotion recognition from body gestures using kinect sensor," in *Communications and Signal Processing (ICCSP), 2014 International Conference on*, April 2014, pp. 056–060.

[71] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, Oct. 2007.

[72] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[73] B. Wang, Z. Chen, and J. Chen, "Gesture recognition by using kinect skeleton tracking system," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference on*, vol. 1, Aug 2013, pp. 418–422.

[74] E. Acorn, N. Dipsis, T. Pincus, and K. Stathis, "Sit-to-stand movement recognition using kinect," in *Statistical Learning and Data Sciences*, ser. Lecture Notes in Computer Science, A. Gammerman, V. Vovk, and H. Papadopoulos, Eds. Springer International Publishing, 2015, vol. 9047, pp. 179–192.

[75] A. Keceli and A. Can, "A multimodal approach for recognizing human actions using depth information," in *Pattern Recognition (ICPR), 2014 22nd International Conference on*, Aug 2014, pp. 421–426.

[76] J. Wu, P. Ishwar, and J. Konrad, "Silhouettes versus skeletons in gesture-based authentication with kinect," in *Advanced Video and Signal Based Surveillance (AVSS), 2014 11th IEEE International Conference on*, Aug 2014, pp. 99–106.

[77] A. Jalal, N. Sarif, J. T. Kim, and T.-S. Kim, "Human activity recognition via

recognized body parts of human depth silhouettes for residents monitoring services at smart home," *Indoor and Built Environment*, vol. 22, no. 1, pp. 271–279, 2013.

[78] A. Kanaujia, N. Kittens, and N. Ramanathan, "Part segmentation of visual hull for 3d human pose estimation," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, June 2013, pp. 542–549.

[79] H. J. Jeon, S. B. Nam, S. U. Park, J. H. Park, C. Yoo, J. T. Kim, and T.-S. Kim, "Unsupervised 3d human pose recognition from a single depth human silhouette using a geodesic map and kinematic body model," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, ser. IMCOM '15. ACM, 2015, pp. 70:1–70:4.

[80] D. D. Luong, S. Lee, and T.-S. Kim, "Human computer interface using the recognized finger parts of hand depth silhouette via random forests," in *Control, Automation and Systems (ICCAS), 2013 13th International Conference on*, Oct 2013, pp. 905–909.

[81] M.-J. Lim, J.-H. Cho, H.-S. Han, and T.-S. Kim, "Upper body pose recognition with labeled depth body parts via random forests and support vector machines," 2013.

[82] AutoDesk, "AutoDesk Character Generator," https://charactergenerator.autodesk.com/, accessed: 9 Jun. 2015.

[83] National University of Singapore (NUS), "Motion Capture Database," http://animation.comp.nus.edu.sg/nusmocap.html, accessed: 9 Jun. 2015.

[84] H. Eidenberger, *Categorization and Machine Learning.* Austria: Atpress, 2014.

[85] T. Kohonen, "Improved versions of learning vector quantization," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, June 1990, pp. 545–550 vol.1.

[86] M. Heidari, G. Seifossadat, and M. Razaz, "Application of decision tree and discrete wavelet transform for an optimized intelligent-based islanding detection method in distributed systems with distributed generations," *Renewable and Sustainable Energy Reviews*, vol. 27, no. 0, pp. 525 – 532, 2013.

[87] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[88] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.

Vitae

- **Name:** Mashaan Awad Salem Alshammari
- **Nationality:** Saudi
- **Date of Birth:** 25/January/1988
- **Email:** `mashaan.awad@outlook.com`
- **Address:** 2373 Az Zahraa 40th street, Ha'il 55481-6372, Saudi Arabia
- **Education:**
  - Received Bachelor of Science (B.Sc.) degree in Computer Science with Second Honors from University of Ha'il (UOH), Ha'il, Saudi Arabia in 2010.
  - Received Master of Science (M.Sc.) degree in Computer Science from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia in 2015.
- **Work Experiences:**
  - **June 2009 - February 2010:** Coop trainee at EXPEC ARC (Exploration and Petroleum Engineering Center, Advanced Research Center), Saudi Aramco, Dhahran, Saudi Arabia.

- **August 2010 - May 2012:** System Engineer at Control Systems and Electrical Department (CSE), Al-Jubail Petrochemical Company (one of SABIC affiliates), Jubail, Saudi Arabia.
- **May 2012 - Present:** Teaching assistant at College of Computer Science and Engineering (CCSE) at University of Ha'il, Ha'il, Saudi Arabia.

- **Publications:**
  - **Alshammari, M.A.**; El-Alfy, E.-S.M., "MapReduce implementation for minimum reduct using parallel genetic algorithm," in *Information and Communication Systems (ICICS), 2015 6th International Conference on*, pp.13-18, 7-9 April 2015
  - Noor, F.; Alhaisoni, M.; **Alshammari, M.A.**; Ramachandran, R.P., "Distinguishing medical drugs from a large set of side effects using a distributed genetic algorithm on a PC cluster," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pp.790-793, 24-27 May 2015